Note to readers:
Please ignore these sidenotes; they're just hints to myself for preparing the index, and they're often flaky!

KNUTH

THE ART OF COMPUTER PROGRAMMING

VOLUME 4 PRE-FASCICLE 6A

A DRAFT OF SECTION 7.2.2.2: SATISFIABILITY

DONALD E. KNUTH *Stanford University*

ADDISON–WESLEY

September 23, 2015

Internet
Stanford GraphBase
MMIX

Internet page http://www-cs-faculty.stanford.edu/~knuth/taocp.html contains current information about this book and related books.

See also http://www-cs-faculty.stanford.edu/~knuth/sgb.html for information about *The Stanford GraphBase*, including downloadable software for dealing with the graphs used in many of the examples in Chapter 7.

See also http://www-cs-faculty.stanford.edu/~knuth/mmixware.html for downloadable software to simulate the MMIX computer.

Copyright © 2015 by Addison–Wesley

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher, except that the official electronic file may be used to print single copies for personal (not commercial) use.

Zeroth printing (revision 5), 15 September 2015

PREFACE

These unforeseen stoppages,
which I own I had no conception of when I first set out;
— but which, I am convinced now, will rather increase than diminish as I advance,
— have struck out a hint which I am resolved to follow;
— and that is, — not to be in a hurry;
— but to go on leisurely, writing and publishing two volumes of my life every year;
— which, if I am suffered to go on quietly, and can make a tolerable bargain
with my bookseller, I shall continue to do as long as I live.

— LAURENCE STERNE, *The Life and Opinions of Tristram Shandy, Gentleman* (1759)

THIS BOOKLET contains draft material that I'm circulating to experts in the field, in hopes that they can help remove its most egregious errors before too many other people see it. I am also, however, posting it on the Internet for courageous and/or random readers who don't mind the risk of reading a few pages that have not yet reached a very mature state. *Beware:* This material has not yet been proofread as thoroughly as the manuscripts of Volumes 1, 2, 3, and 4A were at the time of their first printings. And those carefully-checked volumes, alas, were subsequently found to contain thousands of mistakes.

Given this caveat, I hope that my errors this time will not be so numerous and/or obtrusive that you will be discouraged from reading the material carefully. I did try to make the text both interesting and authoritative, as far as it goes. But the field is vast; I cannot hope to have surrounded it enough to corral it completely. So I beg you to let me know about any deficiencies that you discover.

To put the material in context, this long pre-fascicle contains Section 7.2.2.2 of a long, long chapter on combinatorial algorithms. Chapter 7 will eventually fill at least four volumes (namely Volumes 4A, 4B, 4C, and 4D), assuming that I'm able to remain healthy. It began in Volume 4A with a short review of graph theory and a longer discussion of "Zeros and Ones" (Section 7.1); that volume concluded with Section 7.2.1, "Generating Basic Combinatorial Patterns," which was the first part of Section 7.2, "Generating All Possibilities." Volume 4B will begin with Section 7.2.2, about backtracking in general, and Section 7.2.2.1 will discuss a family of methods called "dancing links," for updating data structures while backtracking. That sets the scene for the present section, which applies those ideas to the important problem of Boolean satisfiability.

* * *

Heule

* * *

After working on this material for more than three years, I've finally gotten most of the major topics into place. Each time I finish drafting a small piece of the final big picture, I've been testing it by adding it to these pages; hence there now are quite a few scraps of text and exercises, which I plan to refine and polish (if the FORCE stays with me).

I hope you will see why I've found this topic to be such a fascinating story; and I hope you'll not get too lost as I move through different parts of the tale. I've tried to explain things in a natural order.

* * *

My notes on combinatorial algorithms have been accumulating for more than fifty years, yet I fear that in many respects my knowledge is woefully behind the times. Please look, for example, at the exercises that I've classed as research problems (rated with difficulty level 46 or higher), namely exercises 5, 39, 112, 193, 194, 236, 283, 516, ...; exercise 223 is also currently unsolved, although I've rated it only '40' because I once thought of an answer (which I have since forgotten!). I've also implicitly mentioned or posed additional unsolved questions in the answers to exercises 18, 19, 68, 84, 105(c,e), 111, 132, 183, 194, 204, 205, 316, 335, 351, 360, 365, 372, 397, 409(c), 476, 480, 486, 487, 488, 501, 511, 515, ... And I still haven't solved exercise 68. Are those problems still open? Please inform me if you know of a solution to any of those intriguing enigmas. And of course if no solution is known today but you do make progress on any of them in the future, I hope you'll let me know.

I urgently need your help also with respect to some exercises that I made up as I was preparing this material. I certainly don't like to receive credit for things that have already been published by others, and most of these results are quite natural "fruits" that were just waiting to be "plucked." Therefore please tell me if you know who deserves to be credited, with respect to the ideas found in exercises 18, 19, 20, 21, 22, 24, 29, 38(b), 62, 63, 65(b), 74, 84(c,d,e), 101, 108, 132, 133, 149, 151, 161, 162, 177, 180, 181, 188, 191, 204(b,c,d), 206, 207, 208, 220, 228, 229, 232, 239, 242, 252, 259, 270, 272, 273, 279, 280, 282, 300, 305, 310, 311, 312, 327, 328, 329, 334, 335, 336(b), 337, 343, 349, 357, 358, 361, 390, 396, 399(c), 404, 406, 410, 411, 414, 419, 423, 427, 432, 433, 435, 439, 462, 463, 464, 465, 470, 472, 473, 475(d,e,f,g), 476, 479, 495, 498, ..., and/or the answers to exercises .... Furthermore I've credited exercise 170 to unpublished work of Heule. Have any of those results ever appeared in print, to your knowledge?

I also wonder if Eq. 7.2.2.2–(169) is "well known."

* * *

Special thanks are due to Armin Biere, Randy Bryant, Sam Buss, Niklas Eén, Ian Gent, Marijn Heule, Holger Hoos, Svante Janson, Peter Jeavons, Daniel Kroening, Oliver Kullmann, Massimo Lauria, Wes Pegden, Will Shortz, Carsten Sinz, Niklas Sörensson, Udo Wermuth, Ryan Williams, and ... for their detailed comments on my early attempts at exposition, as well as to numerous other correspondents who have contributed crucial corrections. Thanks also to Stanford's Information Systems Laboratory for providing extra computer power when my laptop machine was inadequate.

* * *

Wow — Section 7.2.2.2 has turned out to be the longest section, by far, in *The Art of Computer Programming*. The SAT problem is evidently a "killer app," because it is key to the solution of so many other problems. Consequently I can only hope that my lengthy treatment does not also kill off my faithful readers! As I wrote this material, one topic always seemed to flow naturally into another, so there was no neat way to break this section up into separate subsections. (And anyway the format of *TAOCP* doesn't allow for a Section 7.2.2.2.1.)

I've tried to ameliorate the reader's navigation problem by adding subheadings at the top of each right-hand page. Furthermore, as in other sections, the exercises appear in an order that roughly parallels the order in which corresponding topics are taken up in the text. Numerous cross-references are provided between text, exercises, and illustrations, so that you have a fairly good chance of keeping in sync. I've also tried to make the index as comprehensive as possible.

I wrote more than three hundred computer programs while preparing this material, because I find that I don't understand things unless I try to program them. Most of those programs were quite short, of course; but several of them are rather substantial, and possibly of interest to others. Therefore I've made a selection available by listing some of them on the following webpage:

`http://www-cs-faculty.stanford.edu/~knuth/programs.html`

I happily offer a "finder's fee" of $2.56 for each error in this draft when it is first reported to me, whether that error be typographical, technical, or historical. The same reward holds for items that I forgot to put in the index. And valuable suggestions for improvements to the text are worth 32¢ each. (Furthermore, if you find a better solution to an exercise, I'll actually do my best to give you immortal glory, by publishing your name in the eventual book:–)

Volume 4B will begin with a special tutorial and review of probability theory, in an unnumbered section entitled "Mathematical Preliminaries Redux." References to its equations and exercises use the abbreviation 'MPR'. (Think of the word "improvement.") A preliminary version of that section can be found online, as pre-fascicle 5a to Volume 4, if you knew how to find this one.

Cross references to yet-unwritten material sometimes appear as '00'; this impossible value is a placeholder for the actual numbers to be supplied later.

Biere
Bryant
Buss
Eén
Gent
Heule
Hoos
Janson
Jeavons
Kroening
Kullmann
Lauria
Pegden
Shortz
Sinz
Sörensson
Wermuth
Williams
Internet
MPR
Internet

The notational conventions that I've used in the mathematical formulas of this section are summarized either in the Index to Notations of Volume 4A (Appendix B on pages 822–827) or under the heading 'Notational conventions' in the index below.

notational conventions
Knuth

Happy reading!

Stanford, California D. E. K.
69 Umbruary 2015

Here's an exercise for Section 7.2.2.1 that I plan to put eventually into fascicle 5:

Langford pairs
exact cover problem
symmetry breaking
DEFOE
Crusoe

00. [*20*] The problem of Langford pairs on $\{1,1,\ldots,n,n\}$ can be represented as an exact cover problem using columns $\{d_1,\ldots,d_n\}\cup\{s_1,\ldots,s_{2n}\}$; the rows are $d_i\ s_j\ s_k$ for $1\le i\le n$ and $1\le j<k\le 2n$ and $k=i+j+1$, meaning "put digit i into slots j and k."

However, that construction essentially gives us every solution twice, because the left-right reversal of any solution is also a solution. Modify it so that we get only half as many solutions; the others will be the reversals of these.

And here's its cryptic answer (needed in exercise 7.2.2.2–13):

00. Omit the rows with $i=n-[n\text{ even}]$ and $j>n/2$.

(Other solutions are possible. For example, we could omit the rows with $i=1$ and $j\ge n$; that would omit $n-1$ rows instead of only $\lfloor n/2\rfloor$. However, the suggested rule turns out to make the dancing links algorithm run about 10% faster.)

Now I saw, tho' too late, the Folly of
beginning a Work before we count the Cost,
and before we judge rightly of our own Strength to go through with it.
— DANIEL DEFOE, *Robinson Crusoe* (1719)

JOHNSON
Burney

CONTENTS

Chapter 7 — Combinatorial Searching 0

7.2. Generating All Possibilities 0
7.2.1. Generating Basic Combinatorial Patterns 0
7.2.2. Basic Backtrack 0
7.2.2.1. Dancing links 0
7.2.2.2. Satisfiability 1
Example applications 4
Backtracking algorithms 27
Random clauses 47
Resolution of clauses 54
Clause-learning algorithms 60
Monte Carlo algorithms 77
The Local Lemma 81
*Message-passing algorithms 90
*Preprocessing of clauses 95
Encoding constraints into clauses 97
Unit propagation and forcing 103
Symmetry breaking 105
Satisfiability-preserving maps 107
One hundred test cases 113
Tuning the parameters 124
Exploiting parallelism 128
History 129
Exercises 133

Answers to Exercises 185

Index to Algorithms and Theorems 292

Index and Glossary 293

That your book has been delayed I am glad,
since you have gained an opportunity of being more exact.
— SAMUEL JOHNSON, letter to Charles Burney (1 November 1784)

HUME
JAGGER
RICHARDS
Boolean formula
unsatisfiable
satisfying assignment
consistent, see satisfiable
inconsistent, see unsatisfiable
P = NP
NP-complete problem
Knuth
SAT solvers

He reaps no satisfaction but from low and sensual objects,
or from the indulgence of malignant passions.
— DAVID HUME, *The Sceptic* (1742)

I can't get no ...
— MICK JAGGER and KEITH RICHARDS, *Satisfaction* (1965)

7.2.2.2. Satisfiability. We turn now to one of the most fundamental problems of computer science: Given a Boolean formula $F(x_1,\ldots,x_n)$, expressed in so-called "conjunctive normal form" as an AND of ORs, can we "satisfy" F by assigning values to its variables in such a way that $F(x_1,\ldots,x_n)=1$? For example, the formula

$$F(x_1,x_2,x_3)=(x_1\vee\bar{x}_2)\wedge(x_2\vee x_3)\wedge(\bar{x}_1\vee\bar{x}_3)\wedge(\bar{x}_1\vee\bar{x}_2\vee x_3) \qquad (1)$$

is satisfied when $x_1x_2x_3=001$. But if we rule that solution out, by defining

$$G(x_1,x_2,x_3)=F(x_1,x_2,x_3)\wedge(x_1\vee x_2\vee\bar{x}_3), \qquad (2)$$

then G is unsatisfiable: It has no satisfying assignment.

Section 7.1.1 discussed the embarrassing fact that nobody has ever been able to come up with an efficient algorithm to solve the general satisfiability problem, in the sense that the satisfiability of any given formula of size N could be decided in $N^{O(1)}$ steps. Indeed, the famous unsolved question "does P = NP?" is equivalent to asking whether such an algorithm exists. We will see in Section 7.9 that satisfiability is a natural progenitor of every NP-complete problem.*

On the other hand enormous technical breakthroughs in recent years have led to amazingly good ways to approach the satisfiability problem. We now have algorithms that are much more efficient than anyone had dared to believe possible before the year 2000. These so-called "SAT solvers" are able to handle industrial-strength problems, involving millions of variables, with relative ease, and they've had a profound impact on many areas of research such as computer-aided verification. In this section we shall study the principles that underlie modern SAT-solving procedures.

* At the present time very few people believe that P = NP [see *SIGACT News* **43**, 2 (June 2012), 53–77]. In other words, almost everybody who has studied the subject thinks that satisfiability cannot be decided in polynomial time. The author of this book, however, suspects that $N^{O(1)}$-step algorithms do exist, yet that they're unknowable. Almost all polynomial time algorithms are so complicated that they lie beyond human comprehension, and could never be programmed for an actual computer in the real world. Existence is different from embodiment.

To begin, let's define the problem carefully and simplify the notation, so that our discussion will be as efficient as the algorithms that we'll be considering. Throughout this section we shall deal with *variables*, which are elements of any convenient set. Variables are often denoted by x_1, x_2, x_3, ..., as in (1); but any other symbols can also be used, like a, b, c, or even d'''_{74}. We will in fact often use the numerals 1, 2, 3, ... to stand for variables; and in many cases we'll find it convenient to write just j instead of x_j, because it takes less time and less space if we don't have to write so many x's. Thus '2' and 'x_2' will mean the same thing in many of the discussions below.

A *literal* is either a variable or the complement of a variable. In other words, if v is a variable, both v and $\bar{v}$ are literals. If there are n possible variables in some problem, there are $2n$ possible literals. If l is the literal $\bar{x}_2$, which is also written $\bar{2}$, then the complement of l, $\bar{l}$, is x_2, which is also written 2.

The variable that corresponds to a literal l is denoted by $|l|$; thus we have $|v| = |\bar{v}| = v$ for every variable v. Sometimes we write $\pm v$ for a literal that is either v or $\bar{v}$. We might also denote such a literal by σv, where σ is ± 1. The literal l is called *positive* if $|l| = l$; otherwise $|l| = \bar{l}$, and l is said to be *negative*.

Two literals l and l' are *distinct* if $l \neq l'$. They are *strictly distinct* if $|l| \neq |l'|$. A set of literals $\{l_1, \ldots, l_k\}$ is strictly distinct if $|l_i| \neq |l_j|$ for $1 \le i < j \le k$.

The satisfiability problem, like all good problems, can be understood in many equivalent ways, and we will find it convenient to switch from one viewpoint to another as we deal with different aspects of the problem. Example (1) is an AND of clauses, where every clause is an OR of literals; but we might as well regard every clause as simply a *set* of literals, and a formula as a set of clauses. With that simplification, and with 'x_j' identical to 'j', Eq. (1) becomes

$$F = \bigl\{\{1,\bar{2}\},\{2,3\},\{\bar{1},\bar{3}\},\{\bar{1},\bar{2},3\}\bigr\}.$$

And we needn't bother to represent the clauses with braces and commas either; we can simply write out the literals of each clause. With that shorthand we're able to perceive the real essence of (1) and (2):

$$F = \{1\bar{2}, 23, \bar{1}\bar{3}, \bar{1}\bar{2}3\}, \qquad G = F \cup \{12\bar{3}\}. \tag{3}$$

Here F is a set of four clauses, and G is a set of five.

In this guise, the satisfiability problem is equivalent to a *covering problem*, analogous to the exact cover problems that we considered in Section 7.2.2.1: Let

$$T_n = \bigl\{\{x_1,\bar{x}_1\},\{x_2,\bar{x}_2\},\ldots,\{x_n,\bar{x}_n\}\bigr\} = \{1\bar{1}, 2\bar{2}, \ldots, n\bar{n}\}. \tag{4}$$

"Given a set $F = \{C_1, \ldots, C_m\}$, where each C_i is a clause and each clause consists of literals based on the variables $\{x_1, \ldots, x_n\}$, find a set L of n literals that 'covers' $F \cup T_n$, in the sense that every clause contains at least one element of L." For example, the set F in (3) is covered by $L = \{\bar{1}, \bar{2}, 3\}$, and so is the set T_3; hence F is satisfiable. The set G is covered by $\{1, \bar{1}, 2\}$ or $\{1, \bar{1}, 3\}$ or $\cdots$ or $\{\bar{2}, 3, \bar{3}\}$, but not by any three literals that also cover T_3; so G is unsatisfiable.

Similarly, a family F of clauses is satisfiable if and only if it can be covered by a set L of *strictly distinct* literals.

variables
literal
notation: $|v|$
positive
negative
distinct
strictly distinct
covering problem
exact cover problems

If F' is any formula obtained from F by complementing one or more variables, it's clear that F' is satisfiable if and only if F is satisfiable. For example, if we replace 1 by $\bar{1}$ and 2 by $\bar{2}$ in (3) we obtain

$$F' = \{\bar{1}2, \bar{2}3, 1\bar{3}, 123\}, \qquad G = F \cup \{\bar{1}\bar{2}\bar{3}\}.$$

In this case F' is *trivially* satisfiable, because each of its clauses contains a positive literal: Every such formula is satisfied by simply letting L be the set of positive literals. Thus the satisfiability problem is the same as the problem of switching signs (or "polarities") so that no all-negative clauses remain.

Another problem equivalent to satisfiability is obtained by going back to the Boolean interpretation in (1) and complementing both sides of the equation. By De Morgan's laws 7.1.1–(11) and (12) we have

$$\overline{F}(x_1, x_2, x_3) = (\bar{x}_1 \wedge x_2) \vee (\bar{x}_2 \wedge \bar{x}_3) \vee (x_1 \wedge x_3) \vee (x_1 \wedge x_2 \wedge \bar{x}_3); \qquad (5)$$

and F is unsatisfiable $\iff F = 0 \iff \overline{F} = 1 \iff \overline{F}$ is a tautology. Consequently F is satisfiable if and only if $\overline{F}$ is not a tautology: The tautology problem and the satisfiability problem are essentially the same.*

Since the satisfiability problem is so important, we simply call it SAT. And instances of the problem such as (1), in which there are no clauses of length greater than 3, are called 3SAT. In general, kSAT is the satisfiability problem restricted to instances where no clause has more than k literals.

Clauses of length 1 are called *unit clauses*, or unary clauses. Binary clauses, similarly, have length 2; then come ternary clauses, quaternary clauses, and so forth. Going the other way, the *empty clause*, or nullary clause, has length 0 and is denoted by ϵ; it is always unsatisfiable. Short clauses are very important in algorithms for SAT, because they are easier to deal with than long clauses. But long clauses aren't necessarily bad; they're much easier to satisfy than the short ones.

A slight technicality arises when we consider clause length: The binary clause $(x_1 \vee \bar{x}_2)$ in (1) is equivalent to the ternary clause $(x_1 \vee x_1 \vee \bar{x}_2)$ as well as to $(x_1 \vee \bar{x}_2 \vee \bar{x}_2)$ and to longer clauses such as $(x_1 \vee x_1 \vee x_1 \vee \bar{x}_2)$; so we can regard it as a clause of *any* length ≥ 2. But when we think of clauses as *sets* of literals rather than ORs of literals, we usually rule out multisets such as $11\bar{2}$ or $1\bar{2}\bar{2}$ that aren't sets; in that sense a binary clause is *not* a special case of a ternary clause. On the other hand, every binary clause $(x \vee y)$ is equivalent to *two* ternary clauses, $(x \vee y \vee z) \wedge (x \vee y \vee \bar{z})$, if z is another variable; and every k-ary clause is equivalent to two $(k+1)$-ary clauses. Therefore we can assume, if we like, that kSAT deals only with clauses whose length is exactly k.

A clause is tautological (always satisfied) if it contains both v and $\bar{v}$ for some variable v. Tautological clauses can be denoted by $\wp$ (see exercise 7.1.4–222). They never affect a satisfiability problem; so we usually assume that the clauses input to a SAT-solving algorithm consist of strictly distinct literals.

When we discussed the 3SAT problem briefly in Section 7.1.1, we took a look at formula 7.1.1–(32), "the shortest interesting formula in 3CNF." In our new

* Strictly speaking, TAUT is coNP-complete, while SAT is NP-complete; see Section 7.9.

trivially SAT
polarities
De Morgan's laws
TAUT
coNP-complete
SAT
kSAT
3SAT
unit clauses
unary clauses
Binary clauses
ternary clauses
empty clause
nullary clause
ϵ
multisets
tautological
$\wp$
strictly distinct literals

shorthand, it consists of the following eight unsatisfiable clauses:

$$R = \{12\bar{3}, 23\bar{4}, 341, 4\bar{1}2, \bar{1}\bar{2}3, \bar{2}\bar{3}4, \bar{3}\bar{4}\bar{1}, \bar{4}1\bar{2}\}. \qquad (6)$$

This set makes an excellent little test case, so we will refer to it frequently below. (The letter R reminds us that it is based on R. L. Rivest's associative block design 6.5–(13).) The first seven clauses of R, namely

$$R' = \{12\bar{3}, 23\bar{4}, 341, 4\bar{1}2, \bar{1}\bar{2}3, \bar{2}\bar{3}4, \bar{3}\bar{4}\bar{1}\}, \qquad (7)$$

also make nice test data; they are satisfied only by choosing the complements of the literals in the omitted clause, namely $\{4, \bar{1}, 2\}$. More precisely, the literals 4, $\bar{1}$, and 2 are necessary and sufficient to cover R'; we can also include either 3 or $\bar{3}$ in the solution. Notice that (6) is symmetric under the cyclic permutation $1 \to 2 \to 3 \to 4 \to \bar{1} \to \bar{2} \to \bar{3} \to \bar{4} \to 1$ of literals; thus, omitting *any* clause of (6) gives a satisfiability problem equivalent to (7).

Rivest
associative block design
equally spaced occurrences
arithmetic progressions
backtracking
lexicographic order
van der Waerden
$W(k_0, \ldots, k_{b-1})$
waerden+

A simple example. SAT solvers are important because an enormous variety of problems can readily be formulated Booleanwise as ANDs of ORs. Let's begin with a little puzzle that leads to an instructive family of example problems: *Find a binary sequence $x_1 \ldots x_8$ that has no three equally spaced* 0*s and no three equally spaced* 1*s.* For example, the sequence 01001011 almost works; but it doesn't qualify, because x_2, x_5, and x_8 are equally spaced 1s.

If we try to solve this puzzle by backtracking manually through all 8-bit sequences in lexicographic order, we see that $x_1x_2 = 00$ forces $x_3 = 1$. Then $x_1x_2x_3x_4x_5x_6x_7 = 0010011$ leaves us with no choice for x_8. A minute or two of further hand calculation reveals that the puzzle has just six solutions, namely

$$00110011,\ 01011010,\ 01100110,\ 10011001,\ 10100101,\ 11001100. \qquad (8)$$

Furthermore it's easy to see that none of these solutions can be extended to a suitable binary sequence of length 9. We conclude that *every binary sequence $x_1 \ldots x_9$ contains three equally spaced* 0*s or three equally spaced* 1*s.*

Notice now that the condition $x_2x_5x_8 \neq 111$ is the same as the Boolean clause $(\bar{x}_2 \vee \bar{x}_5 \vee \bar{x}_8)$, namely $\bar{2}\bar{5}\bar{8}$. Similarly $x_2x_5x_8 \neq 000$ is the same as 258. So we have just verified that the following 32 clauses are unsatisfiable:

$$\begin{gathered} 123, 234, \ldots, 789, 135, 246, \ldots, 579, 147, 258, 369, 159, \\ \bar{1}\bar{2}\bar{3}, \bar{2}\bar{3}\bar{4}, \ldots, \bar{7}\bar{8}\bar{9}, \bar{1}\bar{3}\bar{5}, \bar{2}\bar{4}\bar{6}, \ldots, \bar{5}\bar{7}\bar{9}, \bar{1}\bar{4}\bar{7}, \bar{2}\bar{5}\bar{8}, \bar{3}\bar{6}\bar{9}, \bar{1}\bar{5}\bar{9}. \end{gathered} \qquad (9)$$

This result is a special case of a general fact that holds for any given positive integers j and k: *If n is sufficiently large, every binary sequence $x_1 \ldots x_n$ contains either j equally spaced* 0*s or k equally spaced* 1*s.* The smallest such n is denoted by $W(j,k)$ in honor of B. L. van der Waerden, who proved an even more general result (see exercise 2.3.4.3–6): *If n is sufficiently large, and if $k_0, \ldots, k_{b-1}$ are positive integers, every b-ary sequence $x_1 \ldots x_n$ contains k_a equally spaced a's for some digit a, $0 \le a < b$.* The least such n is $W(k_0, \ldots, k_{b-1})$.

Let us accordingly define the following set of clauses when $j, k, n > 0$:

$$\begin{aligned} \mathit{waerden}(j,k;n) = {} & \bigl\{(x_i \vee x_{i+d} \vee \cdots \vee x_{i+(j-1)d}) \bigm| 1 \le i \le n-(j-1)d,\ d \ge 1\bigr\} \\ & \cup \bigl\{(\bar{x}_i \vee \bar{x}_{i+d} \vee \cdots \vee \bar{x}_{i+(k-1)d}) \bigm| 1 \le i \le n-(k-1)d,\ d \ge 1\bigr\}. \end{aligned} \qquad (10)$$

Chvátal
Kouril
Paul
Kouril
Ahmed
Kullmann
Snevily
monotonic clauses
exact cover problems
dancing links
Langford pairs
symmetry

The 32 clauses in (9) are $waerden(3,3;9)$; and in general $waerden(j,k;n)$ is an appealing instance of SAT, satisfiable if and only if $n < W(j,k)$.

It's obvious that $W(1,k) = k$ and $W(2,k) = 2k - [k\,\text{even}]$; but when j and k exceed 2 the numbers $W(j,k)$ are quite mysterious. We've seen that $W(3,3) = 9$, and the following nontrivial values are currently known:

$k =$	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
$W(3,k) =$	9	18	22	32	46	58	77	97	114	135	160	186	218	238	279	312	349
$W(4,k) =$	18	35	55	73	109	146	309	?	?	?	?	?	?	?	?	?	?
$W(5,k) =$	22	55	178	206	260	?	?	?	?	?	?	?	?	?	?	?	?
$W(6,k) =$	32	73	206	1132	?	?	?	?	?	?	?	?	?	?	?	?	?

V. Chvátal inaugurated the study of $W(j,k)$ by computing the values for $j+k \le 9$ as well as $W(3,7)$ [*Combinatorial Structures and Their Applications* (1970), 31–33]. Most of the large values in this table have been calculated by state-of-the-art SAT solvers [see M. Kouril and J. L. Paul, *Experimental Math.* **17** (2008), 53–61; M. Kouril, *Integers* **12** (2012), A46:1–A46:13]. The table entries for $j = 3$ suggest that we might have $W(3,k) < k^2$ when $k > 4$, but that isn't true: SAT solvers have also been used to establish the lower bounds

$k =$	20	21	22	23	24	25	26	27	28	29	30
$W(3,k) \ge$	389	416	464	516	593	656	727	770	827	868	903

(which might in fact be the true values for this range of k); see T. Ahmed, O. Kullmann, and H. Snevily [*Discrete Applied Math.* **174** (2014), 27–51].

Notice that the literals in every clause of $waerden(j,k;n)$ have the same sign: They're either all positive or all negative. Does this "monotonic" property make the SAT problem any easier? Unfortunately, no: Exercise 10 proves that *any* set of clauses can be converted to an equivalent set of monotonic clauses.

Exact covering. The exact cover problems that we solved with "dancing links" in Section 7.2.2.1 can easily be reformulated as instances of SAT and handed off to SAT solvers. For example, let's look again at Langford pairs, the task of placing two 1s, two 2s, ..., two n's into $2n$ slots so that exactly k slots intervene between the two appearances of k, for each k. The corresponding exact cover problem when $n = 3$ has nine columns and eight rows (see 7.2.2.1–(00)):

$$d_1\,s_1\,s_3,\ d_1\,s_2\,s_4,\ d_1\,s_3\,s_5,\ d_1\,s_4\,s_6,\ d_2\,s_1\,s_4,\ d_2\,s_2\,s_5,\ d_2\,s_3\,s_6,\ d_3\,s_1\,s_5. \qquad (11)$$

The columns are d_i for $1 \le i \le 3$ and s_j for $1 \le j \le 6$; the row '$d_i\,s_j\,s_k$' means that digit i is placed in slots j and k. Left-right symmetry allows us to omit the row '$d_3\,s_2\,s_6$' from this specification.

We want to select rows of (11) so that each column appears just once. Let the Boolean variable x_j mean 'select row j', for $1 \le j \le 8$; the problem is then to satisfy the nine constraints

$$\begin{aligned} &S_1(x_1,x_2,x_3,x_4) \wedge S_1(x_5,x_6,x_7) \wedge S_1(x_8) \\ &\qquad \wedge S_1(x_1,x_5,x_8) \wedge S_1(x_2,x_6) \wedge S_1(x_1,x_3,x_7) \\ &\qquad\qquad \wedge S_1(x_2,x_4,x_5) \wedge S_1(x_3,x_6,x_8) \wedge S_1(x_4,x_7), \end{aligned} \qquad (12)$$

one for each column. (Here, as usual, $S_1(y_1, \ldots, y_p)$ denotes the symmetric function $[y_1 + \cdots + y_p = 1]$.) For example, we must have $x_5 + x_6 + x_7 = 1$, because column d_2 appears in rows 5, 6, and 7 of (11).

One of the simplest ways to express the symmetric Boolean function S_1 as an AND of ORs is to use $1 + \binom{p}{2}$ clauses:

$$S_1(y_1, \ldots, y_p) = (y_1 \vee \cdots \vee y_p) \wedge \bigwedge_{1 \le j < k \le p} (\bar{y}_j \vee \bar{y}_k). \qquad (13)$$

"At least one of the y's is true, but not two." Then (12) becomes, in shorthand,

$$\{1234, \bar{1}\bar{2}, \bar{1}\bar{3}, \bar{1}\bar{4}, \bar{2}\bar{3}, \bar{2}\bar{4}, \bar{3}\bar{4}, 567, \bar{5}\bar{6}, \bar{5}\bar{7}, \bar{6}\bar{7}, 8,$$
$$158, \bar{1}\bar{5}, \bar{1}\bar{8}, \bar{5}\bar{8}, 26, \bar{2}\bar{6}, 137, \bar{1}\bar{3}, \bar{1}\bar{7}, \bar{3}\bar{7},$$
$$245, \bar{2}\bar{4}, \bar{2}\bar{5}, \bar{4}\bar{5}, 368, \bar{3}\bar{6}, \bar{3}\bar{8}, \bar{6}\bar{8}, 47, \bar{4}\bar{7}\}; \qquad (14)$$

we shall call these clauses *langford*(3). (Notice that only 30 of them are actually distinct, because $\bar{1}\bar{3}$ and $\bar{2}\bar{4}$ appear twice.) Exercise 13 defines *langford*(n); we know from exercise 7–1 that *langford*(n) is satisfiable $\iff n \bmod 4 = 0$ or 3.

The unary clause 8 in (14) tells us immediately that $x_8 = 1$. Then from the binary clauses $\bar{1}\bar{8}$, $\bar{5}\bar{8}$, $\bar{3}\bar{8}$, $\bar{6}\bar{8}$ we have $x_1 = x_5 = x_3 = x_6 = 0$. The ternary clause 137 then implies $x_7 = 1$; finally $x_4 = 0$ (from $\bar{4}\bar{7}$) and $x_2 = 1$ (from 1234). Rows 8, 7, and 2 of (11) now give us the desired Langford pairing 3 1 2 1 3 2.

Incidentally, the function $S_1(y_1, y_2, y_3, y_4, y_5)$ can also be expressed as

$$(y_1 \vee y_2 \vee y_3 \vee y_4 \vee y_5) \wedge (\bar{y}_1 \vee \bar{y}_2) \wedge (\bar{y}_1 \vee \bar{y}_3) \wedge (\bar{y}_1 \vee \bar{t})$$
$$\wedge (\bar{y}_2 \vee \bar{y}_3) \wedge (\bar{y}_2 \vee \bar{t}) \wedge (\bar{y}_3 \vee \bar{t}) \wedge (t \vee \bar{y}_4) \wedge (t \vee \bar{y}_5) \wedge (\bar{y}_4 \vee \bar{y}_5),$$

where t is a new variable. In general, if p gets big, it's possible to express $S_1(y_1, \ldots, y_p)$ with only $3p - 5$ clauses instead of $\binom{p}{2} + 1$, by using $\lfloor (p-3)/2 \rfloor$ new variables as explained in exercise 12. When this alternative encoding is used to represent Langford pairs of order n, we'll call the resulting clauses *langford*′(n).

Do SAT solvers do a better job with the clauses *langford*(n) or *langford*′(n)? Stay tuned: We'll find out later.

Coloring a graph. The classical problem of coloring a graph with at most d colors is another rich source of benchmark examples for SAT solvers. If the graph has n vertices V, we can introduce nd variables v_j, for $v \in V$ and $1 \le j \le d$, signifying that v has color j; the resulting clauses are quite simple:

$$(v_1 \vee v_2 \vee \cdots \vee v_d) \text{ for } v \in V \text{ ("every vertex has at least one color");} \qquad (15)$$
$$(\bar{u}_j \vee \bar{v}_j) \text{ for } u \mathbin{—} v,\ 1 \le j \le d \text{ ("adjacent vertices have different colors").} \qquad (16)$$

We could also add $n\binom{d}{2}$ additional so-called *exclusion clauses*

$$(\bar{v}_i \vee \bar{v}_j) \text{ for } v \in V,\ 1 \le i < j \le d \text{ ("every vertex has at most one color");} \qquad (17)$$

but they're optional, because vertices with more than one color are harmless. Indeed, if we find a solution with $v_1 = v_2 = 1$, we'll be extra happy, because it gives us *two* legal ways to color vertex v. (See exercise 14.)

$S_1(y_1, \ldots, y_p)$
symmetric Boolean function
langford(n)
unary clause
binary clauses
ternary clause
encoding
Langford pairs
langford′(n)
coloring a graph
exclusion clauses
at-most-one

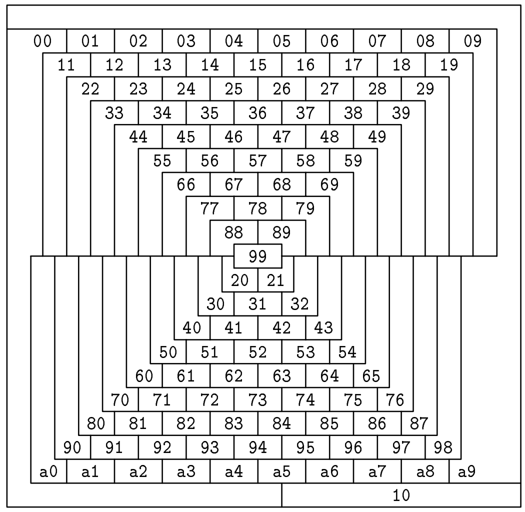


Fig. 33. The McGregor graph of order 10. Each region of this "map" is identified by a two-digit hexadecimal code. Can you color the regions with four colors, never using the same color for two adjacent regions?

Gardner
chess
da Vinci
Ripoff
April Fool
Four Color Theorem
McGregor
Bryant
independent

Martin Gardner astonished the world in 1975 when he reported [*Scientific American* **232**,4 (April 1975), 126–130] that a proper coloring of the planar map in Fig. 33 requires *five* distinct colors, thereby disproving the longstanding four-color conjecture. (In that same column he also cited several other "facts" supposedly discovered in 1974: (i) $e^{\pi\sqrt{163}}$ is an integer; (ii) pawn-to-king-rook-4 ('h4') is a winning first move in chess; (iii) the theory of special relativity is fatally flawed; (iv) Leonardo da Vinci invented the flush toilet; and (v) Robert Ripoff devised a motor that is powered entirely by psychic energy. Thousands of readers failed to notice that they had been April Fooled!)

The map in Fig. 33 actually *can* be 4-colored; you are hereby challenged to discover a suitable way to do this, before turning to the answer of exercise 18. Indeed, the four-color conjecture became the Four Color Theorem in 1976, as mentioned in Section 7. Fortunately that result was still unknown in April of 1975; otherwise this interesting graph would probably never have appeared in print. McGregor's graph has 110 vertices (regions) and 324 edges (adjacencies between regions); hence (15) and (16) yield $110 + 1296 = 1406$ clauses on 440 variables, which a modern SAT solver can polish off quickly.

We can also go much further and solve problems that would be extremely difficult by hand. For example, we can add constraints to limit the number of regions that receive a particular color. Randal Bryant exploited this idea in 2010 to discover that there's a four-coloring of Fig. 33 that uses one of the colors only 7 times (see exercise 17). His coloring is, in fact, unique, and it leads to an explicit way to 4-color the McGregor graphs of all orders $n \geq 3$ (exercise 18).

Such additional constraints can be generated in many ways. We could, for instance, append $\binom{110}{8}$ clauses, one for every choice of 8 regions, specifying that those 8 regions aren't all colored 1. But no, we'd better scratch that idea: $\binom{110}{8} = 409{,}705{,}619{,}895$. Even if we restricted ourselves to the 74,792,876,790 sets of 8 regions that are *independent*, we'd be dealing with far too many clauses.

Sinz
Bailleux
Boufkhad
complete binary tree
cardinality constraints
$S_{\le r}(x_1,\ldots,x_n)$
symmetric threshold functions
binary multiplication
multiplication+

An interesting SAT-oriented way to ensure that $x_1+\cdots+x_n$ is at most r, which works well when n and r are rather large, was found by C. Sinz [*LNCS* **3709** (2005), 827–831]. His method introduces $(n-r)r$ new variables s_j^k for $1\le j\le n-r$ and $1\le k\le r$. If F is any satisfiability problem and if we add the $(n-r-1)r+(n-r)(r+1)$ clauses

$$(\bar{s}_j^k \vee s_{j+1}^k), \qquad \text{for } 1\le j<n-r \text{ and } 1\le k\le r, \tag{18}$$
$$(\bar{x}_{j+k} \vee \bar{s}_j^k \vee s_j^{k+1}), \qquad \text{for } 1\le j\le n-r \text{ and } 0\le k\le r, \tag{19}$$

where $\bar{s}_j^k$ is omitted when $k=0$ and s_j^{k+1} is omitted when $k=r$, then the new set of clauses is satisfiable if and only if F is satisfiable with $x_1+\cdots+x_n\le r$. (See exercise 26.) With this scheme we can limit the number of red-colored regions of McGregor's graph to at most 7 by appending 1538 clauses in 721 new variables.

Another way to achieve the same goal, which turns out to be even better, has been proposed by O. Bailleux and Y. Boufkhad [*LNCS* **2833** (2003), 108–122]. Their method is a bit more difficult to describe, but still easy to implement: Consider a complete binary tree that has $n-1$ internal nodes numbered 1 through $n-1$, and n leaves numbered n through $2n-1$; the children of node k, for $1\le k<n$, are nodes $2k$ and $2k+1$ (see 2.3.4.5–(5)). We form new variables b_j^k for $1<k<n$ and $1\le j\le t_k$, where t_k is the minimum of r and the number of leaves below node k. Then the following clauses, explained in exercise 27, do the job:

$$(\bar{b}_i^{2k} \vee \bar{b}_j^{2k+1} \vee b_{i+j}^k), \ \text{for } 0\le i\le t_{2k},\ 0\le j\le t_{2k+1},\ 1\le i+j\le t_k+1,\ 1<k<n; \tag{20}$$
$$(\bar{b}_i^2 \vee \bar{b}_j^3), \qquad \text{for } 0\le i\le t_2,\ 0\le j\le t_3,\ i+j=r+1. \tag{21}$$

In these formulas we let $t_k=1$ and $b_1^k=x_{k-n+1}$ for $n\le k<2n$; all literals $\bar{b}_0^k$ and b_{r+1}^k are to be omitted. Applying (20) and (21) to McGregor's graph, with $n=110$ and $r=7$, yields just 1216 new clauses in 399 new variables.

The same ideas apply when we want to ensure that $x_1+\cdots+x_n$ is *at least* r, because of the identity $S_{\ge r}(x_1,\ldots,x_n)=S_{\le n-r}(\bar{x}_1,\ldots,\bar{x}_n)$. And exercise 30 considers the case of equality, when our goal is to make $x_1+\cdots+x_n=r$. We'll discuss other encodings of such cardinality constraints below.

Factoring integers. Next on our agenda is a family of SAT instances with quite a different flavor. *Given an $(m+n)$-bit binary integer $z=(z_{m+n}\ldots z_2z_1)_2$, do there exist integers $x=(x_m\ldots x_1)_2$ and $y=(y_n\ldots y_1)_2$ such that $z=x\times y$?* For example, if $m=2$ and $n=3$, we want to invert the binary multiplication

$$\begin{array}{rrrrrr} & & & y_3 & y_2 & y_1 \\ & \times & & & x_2 & x_1 \\ \hline & & & a_3 & a_2 & a_1 \\ & & b_3 & b_2 & b_1 & \\ \hline & c_3 & c_2 & c_1 & & \\ \hline z_5 & z_4 & z_3 & z_2 & z_1 & \end{array} \qquad \begin{aligned} (a_3a_2a_1)_2 &= (y_3y_2y_1)_2\times x_1 \\ (b_3b_2b_1)_2 &= (y_3y_2y_1)_2\times x_2 \end{aligned} \qquad \begin{aligned} z_1 &= a_1 \\ (c_1z_2)_2 &= a_2+b_1 \\ (c_2z_3)_2 &= a_3+b_2+c_1 \\ (c_3z_4)_2 &= b_3+c_2 \\ z_5 &= c_3 \end{aligned} \tag{22}$$

when the z bits are given. This problem is satisfiable when $z=21=(10101)_2$, in the sense that suitable binary values x_1, x_2, y_1, y_2, y_3, a_1, a_2, a_3, b_1, b_2, b_3, c_1, c_2, c_3 do satisfy these equations. But it's unsatisfiable when $z=19=(10011)_2$.

Arithmetical calculations like (22) are easily expressed in terms of clauses that can be fed to a SAT solver: We first specify the computation by constructing a Boolean chain, then we encode each step of the chain in terms of a few clauses. One such chain, if we identify a_1 with z_1 and c_3 with z_5, is

$$\begin{array}{llllll} z_1\leftarrow x_1\wedge y_1, & b_1\leftarrow x_2\wedge y_1, & z_2\leftarrow a_2\oplus b_1, & s\leftarrow a_3\oplus b_2, & z_3\leftarrow s\oplus c_1, & z_4\leftarrow b_3\oplus c_2, \\ a_2\leftarrow x_1\wedge y_2, & b_2\leftarrow x_2\wedge y_2, & c_1\leftarrow a_2\wedge b_1, & p\leftarrow a_3\wedge b_2, & q\leftarrow s\wedge c_1, & z_5\leftarrow b_3\wedge c_2, \\ a_3\leftarrow x_1\wedge y_3, & b_3\leftarrow x_2\wedge y_3, & & & c_2\leftarrow p\vee q, & \end{array} \qquad (23)$$

using a "full adder" to compute c_2z_3 and "half adders" to compute c_1z_2 and c_3z_4 (see 7.1.2–(23) and (24)). And that chain is equivalent to the 49 clauses

$$(x_1\vee\bar z_1)\wedge(y_1\vee\bar z_1)\wedge(\bar x_1\vee\bar y_1\vee z_1)\wedge\cdots\wedge(\bar b_3\vee\bar c_2\vee\bar z_4)\wedge(b_3\vee\bar z_5)\wedge(c_2\vee\bar z_5)\wedge(\bar b_3\vee\bar c_2\vee z_5)$$

obtained by expanding the elementary computations according to simple rules:

$$\begin{aligned} t\leftarrow u\wedge v &\text{ becomes } (u\vee\bar t)\wedge(v\vee\bar t)\wedge(\bar u\vee\bar v\vee t);\\ t\leftarrow u\vee v &\text{ becomes } (\bar u\vee t)\wedge(\bar v\vee t)\wedge(u\vee v\vee\bar t); \qquad (24)\\ t\leftarrow u\oplus v &\text{ becomes } (\bar u\vee v\vee t)\wedge(u\vee\bar v\vee t)\wedge(u\vee v\vee\bar t)\wedge(\bar u\vee\bar v\vee\bar t). \end{aligned}$$

To complete the specification of this factoring problem when, say, $z=(10101)_2$, we simply append the unary clauses $(z_5)\wedge(\bar z_4)\wedge(z_3)\wedge(\bar z_2)\wedge(z_1)$.

Logicians have known for a long time that computational steps can readily be expressed as conjunctions of clauses. Rules such as (24) are now called *Tseytin encoding*, after Gregory Tseytin (1966). Our representation of a small five-bit factorization problem in 49+5 clauses may not seem very efficient; but we will see shortly that m-bit by n-bit factorization corresponds to a satisfiability problem with fewer than $6mn$ variables, and fewer than $20mn$ clauses of length 3 or less.

> *Even if the system has hundreds or thousands of formulas, it can be put into conjunctive normal form "piece by piece," without any "multiplying out."*
>
> — MARTIN DAVIS and HILARY PUTNAM (1958)

Suppose $m\le n$. The easiest way to set up Boolean chains for multiplication is probably to use a scheme that goes back to John Napier's *Rabdologiæ* (Edinburgh, 1617), pages 137–143, as modernized by Luigi Dadda [*Alta Frequenza* **34** (1964), 349–356]: First we form all mn products $x_i\wedge y_j$, putting every such bit into $bin[i+j]$, which is one of $m+n$ "bins" that hold bits to be added for a particular power of 2 in the binary number system. The bins will contain respectively $(0,1,2,\ldots,m,m,\ldots,m,\ldots,2,1)$ bits at this point, with $n-m+1$ occurrences of "m" in the middle. Now we look at $bin[k]$ for $k=2,3,\ldots$. If $bin[k]$ contains a single bit b, we simply set $z_{k-1}\leftarrow b$. If it contains two bits $\{b,b'\}$, we use a half adder to compute $z_{k-1}\leftarrow b\oplus b'$, $c\leftarrow b\wedge b'$, and we put the carry bit c into $bin[k+1]$. Otherwise $bin[k]$ contains $t\ge 3$ bits; we choose any three of them, say $\{b,b',b''\}$, and remove them from the bin. With a full adder we then compute $r\leftarrow b\oplus b'\oplus b''$ and $c\leftarrow\langle bb'b''\rangle$, so that $b+b'+b''=r+2c$; and we put r into $bin[k]$, c into $bin[k+1]$. This decreases t by 2, so eventually we will have computed z_{k-1}. Exercise 41 quantifies the exact amount of calculation involved.

Boolean chain
full adder
half adders
unary clauses
Tseytin encoding
conjunctive normal form
DAVIS
PUTNAM
Napier
Dadda
binary number system
half adder
carry bit
full adder
notation $\langle abc\rangle$
median operation
ternary operations

This method of encoding multiplication into clauses is quite flexible, since we're allowed to choose *any* three bits from $bin[k]$ whenever four or more bits are present. We could use a first-in-first-out strategy, always selecting bits from the "rear" and placing their sum at the "front"; or we could work last-in-first-out, essentially treating $bin[k]$ as a stack instead of a queue. We could also select the bits randomly, to see if this makes our SAT solver any happier. Later in this section we'll refer to the clauses that represent the factoring problem by calling them $factor_fifo(m,n,z)$, $factor_lifo(m,n,z)$, or $factor_rand(m,n,z,s)$, respectively, where s is a seed for the random number generator used to generate them.

It's somewhat mind-boggling to realize that numbers can be factored without using any number theory! No greatest common divisors, no applications of Fermat's theorems, etc., are anywhere in sight. We're providing no hints to the solver except for a bunch of Boolean formulas that operate almost blindly at the bit level. Yet factors are found.

Of course we can't expect this method to compete with the sophisticated factorization algorithms of Section 4.5.4. But the problem of factoring does demonstrate the great versatility of clauses. And its clauses can be combined with *other* constraints that go well beyond any of the problems we've studied before.

Fault testing. Lots of things can go wrong when computer chips are manufactured in the "real world," so engineers have long been interested in constructing test patterns to check the validity of a particular circuit. For example, suppose that all but one of the logical elements are functioning properly in some chip; the bad one, however, is stuck: Its output is constant, always the same regardless of the inputs that it is given. Such a failure is called a *single-stuck-at fault*.

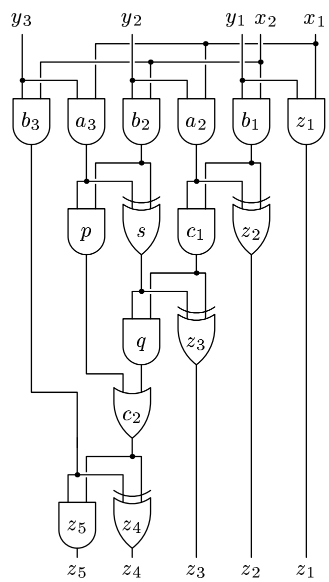


Fig. 34. A circuit that corresponds to (23).

Figure 34 illustrates a typical digital circuit in detail: It implements the 15 Boolean operations of (23) as a network that produces five output signals $z_5z_4z_3z_2z_1$ from the five inputs $y_3y_2y_1x_2x_1$. In addition to having 15 AND, OR, and XOR gates, each of which transforms two inputs into one output, it has 15 "fanout" gates (indicated by dots at junction points), each of which splits one input into two outputs. As a result it comprises 50 potentially distinct logical signals, one for each internal "wire." Exercise 47 shows that a circuit with m outputs, n inputs, and g conventional 2-to-1 gates will have $g+m-n$ fanout gates and $3g+2m-n$ wires. A circuit with w wires has $2w$ possible single-stuck-at faults, namely w faults in which the signal on a wire is stuck at 0 and w more on which it is stuck at 1.

Table 1 shows 101 scenarios that are possible when the 50 wires of Fig. 34 are activated by one particular sequence of inputs, assuming that at

first-in-first-out
FIFO: first in first out
last-in-first-out
stack
queue
$factor_fifo(m,n,z)$
$factor_lifo(m,n,z)$
$factor_rand(m,n,z,s)$
Fermat
Fault testing
ATPG: Automatic test pattern generation, see F
test patterns
circuit
circuit: see also Boolean chain
single-stuck-at fault
fanout

default
bitwise

Table 1

SINGLE-STUCK-AT FAULTS IN FIGURE 34 WHEN $x_2x_1 = 11$, $y_3y_2y_1 = 110$

	OK	x_1	x_1^1	x_1^2	x_1^3	x_1^4	x_2	x_2^1	x_2^2	x_2^3	x_2^4	y_1	y_1^1	y_1^2	y_2	y_2^1	y_2^2	y_3	y_3^1	y_3^2	z_1	a_2	a_2^1	a_2^2	a_3	a_3^1	a_3^2	b_1	b_1^1	b_1^2	b_2	b_2^1	b_2^2	b_3	b_3^1	b_3^2	z_2	c_1	c_1^1	c_1^2	s	s^1	s^2	p	z_3	q	c_2	c_2^1	c_2^2	z_4	z_5
$x_1 \leftarrow$ input	1	**01**	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
$x_1^1 \leftarrow x_1$	1	01	**01**	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
$x_1^2 \leftarrow x_1$	1	01	11	**01**	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
$x_1^3 \leftarrow x_1^1$	1	01	01	11	**01**	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
$x_1^4 \leftarrow x_1^1$	1	01	01	11	11	**01**	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
$x_2 \leftarrow$ input	1	11	11	11	11	11	**01**	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
$x_2^1 \leftarrow x_2$	1	11	11	11	11	11	01	**01**	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
$x_2^2 \leftarrow x_2$	1	11	11	11	11	11	01	11	**01**	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
$x_2^3 \leftarrow x_2^1$	1	11	11	11	11	11	01	01	11	**01**	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
$x_2^4 \leftarrow x_2^1$	1	11	11	11	11	11	01	01	11	11	**01**	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
$y_1 \leftarrow$ input	0	00	00	00	00	00	00	00	00	00	00	**01**	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$y_1^1 \leftarrow y_1$	0	00	00	00	00	00	00	00	00	00	00	01	**01**	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$y_1^2 \leftarrow y_1$	0	00	00	00	00	00	00	00	00	00	00	01	00	**01**	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$y_2 \leftarrow$ input	1	11	11	11	11	11	11	11	11	11	11	11	11	11	**01**	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
$y_2^1 \leftarrow y_2$	1	11	11	11	11	11	11	11	11	11	11	11	11	11	01	**01**	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
$y_2^2 \leftarrow y_2$	1	11	11	11	11	11	11	11	11	11	11	11	11	11	01	11	**01**	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
$y_3 \leftarrow$ input	1	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	**01**	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
$y_3^1 \leftarrow y_3$	1	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	01	**01**	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
$y_3^2 \leftarrow y_3$	1	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	01	11	**01**	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
$z_1 \leftarrow x_1^2 \wedge y_1^1$	0	00	00	00	00	00	00	00	00	00	00	01	01	00	00	00	00	00	00	00	**01**	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$a_2 \leftarrow x_1^3 \wedge y_2^1$	1	01	01	11	01	11	11	11	11	11	11	11	11	11	01	01	11	11	11	11	11	**01**	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
$a_2^1 \leftarrow a_2$	1	01	01	11	01	11	11	11	11	11	11	11	11	11	01	01	11	11	11	11	11	01	**01**	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
$a_2^2 \leftarrow a_2$	1	01	01	11	01	11	11	11	11	11	11	11	11	11	01	01	11	11	11	11	11	01	11	**01**	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
$a_3 \leftarrow x_1^4 \wedge y_3^1$	1	01	01	11	11	01	11	11	11	11	11	11	11	11	11	11	11	01	01	11	11	11	11	11	**01**	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
$a_3^1 \leftarrow a_3$	1	01	01	11	11	01	11	11	11	11	11	11	11	11	11	11	11	01	01	11	11	11	11	11	01	**01**	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
$a_3^2 \leftarrow a_3$	1	01	01	11	11	01	11	11	11	11	11	11	11	11	11	11	11	01	01	11	11	11	11	11	01	11	**01**	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
$b_1 \leftarrow x_2^2 \wedge y_1^2$	0	00	00	00	00	00	00	00	00	00	00	01	00	01	00	00	00	00	00	00	00	00	00	00	00	00	00	**01**	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$b_1^1 \leftarrow b_1$	0	00	00	00	00	00	00	00	00	00	00	01	00	01	00	00	00	00	00	00	00	00	00	00	00	00	00	01	**01**	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$b_1^2 \leftarrow b_1$	0	00	00	00	00	00	00	00	00	00	00	01	00	01	00	00	00	00	00	00	00	00	00	00	00	00	00	01	00	**01**	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$b_2 \leftarrow x_2^3 \wedge y_2^2$	1	11	11	11	11	11	01	01	11	01	11	11	11	11	01	11	01	11	11	11	11	11	11	11	11	11	11	11	11	11	**01**	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
$b_2^1 \leftarrow b_2$	1	11	11	11	11	11	01	01	11	01	11	11	11	11	01	11	01	11	11	11	11	11	11	11	11	11	11	11	11	11	01	**01**	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
$b_2^2 \leftarrow b_2$	1	11	11	11	11	11	01	01	11	01	11	11	11	11	01	11	01	11	11	11	11	11	11	11	11	11	11	11	11	11	01	11	**01**	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
$b_3 \leftarrow x_2^4 \wedge y_3^2$	1	11	11	11	11	11	01	01	11	11	01	11	11	11	11	11	11	01	11	01	11	11	11	11	11	11	11	11	11	11	11	11	11	**01**	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
$b_3^1 \leftarrow b_3$	1	11	11	11	11	11	01	01	11	11	01	11	11	11	11	11	11	01	11	01	11	11	11	11	11	11	11	11	11	11	11	11	11	01	**01**	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
$b_3^2 \leftarrow b_3$	1	11	11	11	11	11	01	01	11	11	01	11	11	11	11	11	11	01	11	01	11	11	11	11	11	11	11	11	11	11	11	11	11	01	11	**01**	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
$z_2 \leftarrow a_2^1 \oplus b_1^1$	1	01	01	11	01	11	11	11	11	11	11	10	11	10	01	01	11	11	11	11	11	01	01	11	11	11	11	10	10	11	11	11	11	11	11	11	**01**	11	11	11	11	11	11	11	11	11	11	11	11	11	11
$c_1 \leftarrow a_2^2 \wedge b_1^2$	0	00	00	00	00	00	00	00	00	00	00	01	00	01	00	00	00	00	00	00	00	00	00	00	00	00	00	01	00	01	00	00	00	00	00	00	00	**01**	00	00	00	00	00	00	00	00	00	00	00	00	00
$c_1^1 \leftarrow c_1$	0	00	00	00	00	00	00	00	00	00	00	01	00	01	00	00	00	00	00	00	00	00	00	00	00	00	00	01	00	01	00	00	00	00	00	00	00	01	**01**	00	00	00	00	00	00	00	00	00	00	00	00
$c_1^2 \leftarrow c_1$	0	00	00	00	00	00	00	00	00	00	00	01	00	01	00	00	00	00	00	00	00	00	00	00	00	00	00	01	00	01	00	00	00	00	00	00	00	01	00	**01**	00	00	00	00	00	00	00	00	00	00	00
$s \leftarrow a_3^1 \oplus b_2^1$	0	10	10	00	00	10	10	10	00	10	00	00	00	00	10	00	10	10	10	00	00	00	00	00	10	10	00	00	00	00	10	10	00	00	00	00	00	00	00	00	**01**	00	00	00	00	00	00	00	00	00	00
$s^1 \leftarrow s$	0	10	10	00	00	10	10	10	00	10	00	00	00	00	10	00	10	10	10	00	00	00	00	00	10	10	00	00	00	00	10	10	00	00	00	00	00	00	00	00	01	**01**	00	00	00	00	00	00	00	00	00
$s^2 \leftarrow s$	0	10	10	00	00	10	10	10	00	10	00	00	00	00	10	00	10	10	10	00	00	00	00	00	10	10	00	00	00	00	10	10	00	00	00	00	00	00	00	00	01	00	**01**	00	00	00	00	00	00	00	00
$p \leftarrow a_3^2 \wedge b_2^2$	1	01	01	11	11	01	01	01	11	01	11	11	11	11	01	11	01	01	01	11	11	11	11	11	01	11	01	11	11	11	01	11	01	11	11	11	11	11	11	11	11	11	11	**01**	11	11	11	11	11	11	11
$z_3 \leftarrow s^1 \oplus c_1^1$	0	10	10	00	00	10	10	10	00	10	00	01	00	01	10	00	10	10	10	00	00	00	00	00	10	10	00	01	00	01	10	10	00	00	00	00	00	01	01	00	01	01	00	00	**01**	00	00	00	00	00	00
$q \leftarrow s^2 \wedge c_1^2$	0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	**01**	00	00	00	00	00
$c_2 \leftarrow p \vee q$	1	01	01	11	11	01	01	01	11	01	11	11	11	11	01	11	01	01	01	11	11	11	11	11	01	11	01	11	11	11	01	11	01	11	11	11	11	11	11	11	11	11	11	01	11	11	**01**	11	11	11	11
$c_2^1 \leftarrow c_2$	1	01	01	11	11	01	01	01	11	01	11	11	11	11	01	11	01	01	01	11	11	11	11	11	01	11	01	11	11	11	01	11	01	11	11	11	11	11	11	11	11	11	11	01	11	11	01	**01**	11	11	11
$c_2^2 \leftarrow c_2$	1	01	01	11	11	01	01	01	11	01	11	11	11	11	01	11	01	01	01	11	11	11	11	11	01	11	01	11	11	11	01	11	01	11	11	11	11	11	11	11	11	11	11	01	11	11	01	11	**01**	11	11
$z_4 \leftarrow b_3^1 \oplus c_2^1$	0	10	10	00	00	10	00	00	00	10	10	00	00	00	10	00	10	00	10	10	00	00	00	00	10	00	10	00	00	00	10	00	10	10	10	00	00	00	00	00	00	00	00	10	00	00	10	10	00	**01**	00
$z_5 \leftarrow b_3^2 \wedge c_2^2$	1	01	01	11	11	01	01	01	11	01	01	11	11	11	01	11	01	01	01	01	11	11	11	11	01	11	01	11	11	11	01	11	01	01	11	01	11	11	11	11	11	11	11	01	11	11	01	11	01	11	**01**

most one stuck-at fault is present. The column headed OK shows the correct behavior of the Boolean chain (which nicely multiplies $x = 3$ by $y = 6$ and obtains $z = 18$). We can call these the "default" values, because, well, they have no faults. The other 100 columns show what happens if all but one of the 50 wires have error-free signals; the two columns under b_2^1, for example, illustrate the results when the rightmost wire that fans out from gate b_2 is stuck at 0 or 1. Each row is obtained bitwise from previous rows or inputs, except that the boldface digits are forced. When a boldface value agrees with the default, its entire column is correct; otherwise errors might propagate. All values above the bold diagonal match the defaults.

If we want to test a chip that has n inputs and m outputs, we're allowed to apply test patterns to the inputs and see what outputs are produced. Close

inspection shows, for instance, that the pattern considered in Table 1 doesn't detect an error when q is stuck at 1, even though q should be 0, because all five output bits $z_5z_4z_3z_2z_1$ are correct in spite of that error. In fact, the value of $c_1 \leftarrow p \vee q$ is unaffected by a bad q, because $p = 1$ in this example. Similarly, the fault "x_1^2 stuck at 0" doesn't propagate into $z_1 \leftarrow x_1^2 \wedge y_1^1$ because $y_1^1 = 0$. Altogether 44 faults, not 50, are discovered by this particular test pattern.

All of the relevant repeatable faults, whether they're single-stuck-at or wildly complicated, could obviously be discovered by testing all 2^n possible patterns. But that's out of the question unless n is quite small. Fortunately, testing isn't hopeless, because satisfactory results are usually obtained in practice if we do have enough test patterns to detect all of the detectable single-stuck-at faults. Exercise 49 shows that just five patterns suffice to certify Fig. 34 by this criterion.

The detailed analysis in exercise 49 also shows, surprisingly, that one of the faults, namely "s^2 stuck at 1," cannot be detected! Indeed, an erroneous s^2 can propagate to an erroneous q only if $c_1^2 = 1$, and that forces $x_1 = x_2 = y_1 = y_2 = 1$; only two possibilities remain, and neither $y_3 = 0$ nor $y_3 = 1$ reveals the fault. Consequently we can simplify the circuit by removing gate q; the chain (23) becomes shorter, with "$q \leftarrow s \wedge c_1$, $c_2 \leftarrow p \vee q$" replaced by "$c_2 \leftarrow p \vee c_1$."

Of course Fig. 34 is just a tiny little circuit, intended only to introduce the concept of stuck-at faults. Test patterns are needed for the much larger circuits that arise in real computers; and we will see that SAT solvers can help us to find them. Consider, for example, the generic multiplier circuit *prod*(m, n), which is part of the Stanford GraphBase. It multiplies an m-bit number x by an n-bit number y, producing an $(m + n)$-bit product z. Furthermore, it's a so-called "parallel multiplier," with delay time $O(\log(m + n))$; thus it's much more suited to hardware design than methods like the *factor_fifo* schemes that we considered above, because those circuits need $\Omega(m + n)$ time for carries to propagate.

Let's try to find test patterns that will smoke out all of the single-stuck-at faults in *prod*$(32, 32)$, which is a circuit of depth 33 that has 64 inputs, 64 outputs, 3660 AND gates, 1203 OR gates, 2145 XOR gates, and (therefore) 7008 fanout gates and 21,088 wires. How can we guard it against 42,176 different faults?

Before we construct clauses to facilitate that task, we should realize that most of the single-stuck-at faults are easily detected by choosing patterns at random, since faults usually cause big trouble and are hard to miss. Indeed, choosing x = #3243F6A8 and y = #885A308D more or less at random already eliminates 14,733 cases; and (x, y) = (#2B7E1516, #28AED2A6) eliminates 6,918 more. We might as well keep doing this, because bitwise operations such as those in Table 1 are fast. Experience with the smaller multiplier in Fig. 34 suggests that we get more effective tests if we bias the inputs, choosing each bit to be 1 with probability .9 instead of .5 (see exercise 49). A million such random inputs will then generate, say, 243 patterns that detect all but 140 of the faults.

Our remaining job, then, is essentially to find 140 needles in a haystack of size 2^{64}, after having picked $42{,}176 - 140 = 42{,}036$ pieces of low-hanging fruit. And that's where a SAT solver is useful. Consider, for example, the analogous but simpler problem of finding a test pattern for "q stuck at 0" in Fig. 34.

Boolean evaluation
prod
Stanford GraphBase
multiplies
parallel multiplier
factor_fifo
fanout gates
wires
random
pi, as source of random data
e, as source of random data
bitwise operations

We can use the 49 clauses F derived from (23) to represent the well-behaved circuit; and we can imagine corresponding clauses F' that represent the faulty computation, using "primed" variables z'_1, a'_2, $\ldots$, z'_5. Thus F' begins with $(x_1 \vee \bar{z}'_1) \wedge (y_1 \vee \bar{z}'_1)$ and ends with $(\bar{b}'_3 \vee \bar{c}'_2 \vee z'_5)$; it's like F except that the clauses representing $q' \leftarrow s' \wedge c'_1$ in (23) are changed to simply $\bar{q}'$ (meaning that q' is stuck at 0). Then the clauses of F and F', together with a few more clauses to state that $z_1 \neq z'_1$ or $\cdots$ or $z_5 \neq z'_5$, will be satisfiable only by variables for which $(y_3y_2y_1)_2 \times (x_2x_1)_2$ is a suitable test pattern for the given fault.

This construction of F' can obviously be simplified, because z'_1 is identical to z_1; any signal that differs from the correct value must be located "downstream" from the one-and-only fault. Let's say that a wire is *tarnished* if it is the faulty wire or if at least one of its input wires is tarnished. We introduce new variables g' only for wires g that are tarnished. Thus, in our example, the only clauses F' that are needed to extend F to a faulty companion circuit are $\bar{q}'$ and the clauses that correspond to $c'_2 \leftarrow p \vee q'$, $z'_4 \leftarrow b_3 \oplus c'_2$, $z'_5 \leftarrow b_3 \wedge c'_2$.

Moreover, any fault that is revealed by a test pattern must have an *active path* of wires, leading from the fault to an output; all wires on this path must carry a faulty signal. Therefore Tracy Larrabee [*IEEE Trans.* **CAD-11** (1992), 4–15] decided to introduce additional "sharped" variables $g^\sharp$ for each tarnished wire, meaning that g lies on the active path. The two clauses

$$(\bar{g}^\sharp \vee g \vee g') \;\wedge\; (\bar{g}^\sharp \vee \bar{g} \vee \bar{g}') \qquad (25)$$

ensure that $g \neq g'$ whenever g is part of that path. Furthermore we have $(\bar{v}^\sharp \vee g^\sharp)$ whenever g is an AND, OR, or XOR gate with tarnished input v. Fanout gates are slightly tricky in this regard: When wires g^1 and g^2 fan out from a tarnished wire g, we need variables $g^{1\sharp}$ and $g^{2\sharp}$ as well as $g^\sharp$; and we introduce the clause

$$(\bar{g}^\sharp \vee g^{1\sharp} \vee g^{2\sharp}) \qquad (26)$$

to specify that the active path takes at least one of the two branches.

According to these rules, our example acquires the new variables $q^\sharp$, $c_2^\sharp$, $c_2^{1\sharp}$, $c_2^{2\sharp}$, $z_4^\sharp$, $z_5^\sharp$, and the new clauses

$$(\bar{q}^\sharp \vee q \vee q') \wedge (\bar{q}^\sharp \vee \bar{q} \vee \bar{q}') \wedge (\bar{q}^\sharp \vee c_2^\sharp) \wedge (\bar{c}_2^\sharp \vee c_2 \vee c'_2) \wedge (\bar{c}_2^\sharp \vee \bar{c}_2 \vee \bar{c}'_2) \wedge (\bar{c}_2^\sharp \vee c_2^{1\sharp} \vee c_2^{2\sharp}) \wedge$$
$$(\bar{c}_2^{1\sharp} \vee z_4^\sharp) \wedge (\bar{z}_4^\sharp \vee z_4 \vee z'_4) \wedge (\bar{z}_4^\sharp \vee \bar{z}_4 \vee \bar{z}'_4) \wedge (\bar{c}_2^{2\sharp} \vee z_5^\sharp) \wedge (\bar{z}_5^\sharp \vee z_5 \vee z'_5) \wedge (\bar{z}_5^\sharp \vee \bar{z}_5 \vee \bar{z}'_5).$$

The active path begins at q, so we assert the unit clause $(q^\sharp)$; it ends at a tarnished output, so we also assert $(z_4^\sharp \vee z_5^\sharp)$. The resulting set of clauses will find a test pattern for this fault if and only if the fault is detectable. Larrabee found that such active-path variables provide important clues to a SAT solver and significantly speed up the solution process.

Returning to the large circuit *prod*(32, 32), one of the 140 hard-to-test faults is "W_{21}^{26} stuck at 1," where W_{21}^{26} denotes the 26th extra wire that fans out from the OR gate called W_{21} in §75 of the Stanford GraphBase program GB_GATES; W_{21}^{26} is an input to gate $b_{40}^{40} \leftarrow d_{40}^{19} \wedge W_{21}^{26}$ in §80 of that program. Test patterns for that fault can be characterized by a set of 23,194 clauses in 7,082 variables

tarnished
active path
Larrabee
AND
OR
XOR
Fanout gates
unit clause
SAT solver
Stanford GraphBase
GB_GATES

Knuth
GB_GATES
number theory
Aurifeuillian
learning a Boolean function–
Boolean function–
DNF
disjunctive normal form

(of which only 4 variables are "primed" and 4 are "sharped"). Fortunately the solution $(x, y) = ({}^{\#}\mathtt{7F13FEDD}, {}^{\#}\mathtt{5FE57FFE})$ was found rather quickly in the author's experiments; and this pattern also killed off 13 of the other cases, so the score was now "14 down and 126 to go"!

The next fault sought was "$A_5^{36,2}$ stuck at 1," where $A_5^{36,2}$ is the second extra wire to fan out from the AND gate A_5^{36} in §72 of GB_GATES (an input to $R_{11}^{36} \leftarrow A_5^{36,2} \wedge R_1^{35,2}$). This fault corresponds to 26,131 clauses on 8,342 variables; but the SAT solver took a quick look at those clauses and decided almost instantly that they are *unsatisfiable*. Therefore the fault is undetectable, and the circuit *prod*$(32, 32)$ can be simplified by setting $R_{11}^{36} \leftarrow R_1^{35,2}$. A closer look showed, in fact, that clauses corresponding to the Boolean equations

$$x = y \wedge z, \quad y = v \wedge w, \quad z = t \wedge u, \quad u = v \oplus w$$

were present (where $t = R_{13}^{44}$, $u = A_{58}^{45}$, $v = R_4^{44}$, $w = A_{14}^{45}$, $x = R_{23}^{46}$, $y = R_{13}^{45}$, $z = R_{19}^{45}$); these clauses *force* $x = 0$. Therefore it was not surprising to find that the list of unresolved faults also included R_{23}^{46}, $R_{23}^{46,1}$ and $R_{23}^{46,2}$ stuck at 0. Altogether 26 of the 140 faults undetected by random inputs turned out to be *absolutely* undetectable; and only one of these, namely "Q_{26}^{46} stuck at 0," required a nontrivial proof of undetectability.

Some of the $126-26 = 100$ faults remaining on the to-do list turned out to be significant challenges for the SAT solver. While waiting, the author therefore had time to take a look at a few of the previously found solutions, and noticed that those patterns themselves were forming a pattern! Sure enough, the extreme portions of this large and complicated circuit actually have a fairly simple structure, stuck-at-fault-wise. Hence number theory came to the rescue: The factorization ${}^{\#}\mathtt{87FBC059} \times {}^{\#}\mathtt{F0F87817} = 2^{63} - 1$ solved many of the toughest challenges, some of which occur with probability less than 2^{-34} when 32-bit numbers are multiplied; and the "Aurifeuillian" factorization $(2^{31} - 2^{16} + 1)(2^{31} + 2^{16} + 1) = 2^{62} + 1$, which the author had known for more than forty years (see Eq. 4.5.4–(15)), polished off most of the others.

The bottom line (see exercise 51) is that all 42,150 of the detectable single-stuck-at faults of the parallel multiplication circuit *prod*$(32, 32)$ can actually be detected with at most 196 well-chosen test patterns.

Learning a Boolean function. Sometimes we're given a "black box" that evaluates a Boolean function $f(x_1, \ldots, x_N)$. We have no way to open the box, but we suspect that the function is actually quite simple. By plugging in various values for $x = x_1 \ldots x_N$, we can observe the box's behavior and possibly learn the hidden rule that lies inside. For example, a secret function of $N = 20$ Boolean variables might take on the values shown in Table 2, which lists 16 cases where $f(x) = 1$ and 16 cases where $f(x) = 0$.

Suppose we assume that the function has a DNF (disjunctive normal form) with only a few terms. We'll see in a moment that it's easy to express such an assumption as a satisfiability problem. And when the author constructed clauses corresponding to Table 2 and presented them to a SAT solver, he did in fact learn

Table 2

VALUES TAKEN ON BY AN UNKNOWN FUNCTION

Cases where $f(x) = 1$

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9					$\cdots$						x_{20}
1	1	0	0	1	0	0	1	0	0	0	0	1	1	1	1	1	1	0	1
1	0	1	0	1	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1
0	1	1	0	1	0	0	0	1	1	0	0	0	0	1	0	0	0	1	1
0	1	0	0	1	1	0	0	0	1	0	0	1	1	0	0	0	1	1	0
0	1	1	0	0	0	1	0	1	0	0	0	1	0	1	1	1	0	0	0
0	0	0	0	1	1	0	1	1	1	0	0	0	0	0	1	1	1	0	0
1	1	0	1	0	0	0	1	0	0	1	0	1	0	0	1	0	0	0	0
0	0	1	0	0	1	0	0	1	1	1	0	0	0	0	0	1	0	0	0
1	0	0	0	1	0	1	0	0	1	1	0	0	1	1	1	1	1	0	0
1	1	0	0	0	1	1	1	0	1	0	0	0	0	0	0	0	0	1	0
0	0	0	0	1	0	1	1	1	0	1	1	1	1	1	0	1	0	1	0
0	1	1	0	0	0	1	1	1	0	1	1	0	0	0	1	0	0	1	1
1	0	0	1	1	0	1	1	0	0	1	0	0	0	1	0	0	1	0	1
0	0	0	1	0	1	0	0	1	0	1	0	0	0	0	0	1	0	0	0
0	1	1	1	1	0	0	1	1	0	0	0	1	1	1	0	0	0	1	1
0	1	0	0	0	0	0	0	0	1	0	0	1	1	0	1	1	1	0	1

Cases where $f(x) = 0$

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9					$\cdots$						x_{20}
1	0	1	0	1	1	0	1	1	1	1	1	1	0	0	0	0	1	0	1
0	1	0	0	0	1	0	1	1	0	0	0	1	0	1	0	0	0	1	0
1	0	1	1	1	0	1	1	0	1	0	0	1	0	1	0	1	0	0	1
1	0	1	0	1	0	1	0	1	1	1	1	1	1	0	1	1	1	0	0
0	1	0	1	0	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0
0	1	1	1	0	0	1	1	1	1	0	1	0	0	1	1	1	1	0	0
1	1	1	1	0	0	0	1	1	1	0	1	1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0	0	1	0	1	1	0	0	0	0	0	1	1
1	1	0	0	1	1	1	0	0	0	1	0	1	1	0	1	0	0	1	1
0	1	1	0	1	0	0	1	0	1	0	1	1	0	1	0	1	0	0	1
1	1	1	0	0	0	0	1	0	0	1	1	0	1	1	0	0	1	0	0
0	0	0	1	0	0	0	1	0	1	0	0	0	1	1	0	0	1	0	0
0	0	1	1	0	0	1	1	1	1	1	1	1	0	1	1	1	1	0	0
1	1	0	0	1	0	0	1	0	0	1	1	1	0	0	1	1	1	0	1
1	1	0	0	1	1	1	0	0	0	1	0	0	1	0	0	1	0	0	1
1	0	1	1	0	0	1	1	1	1	1	0	1	1	1	1	1	0	0	1

almost immediately that a very simple formula is consistent with all of the data:

$$f(x_1,\ldots,x_{20}) = \bar{x}_2\bar{x}_3\bar{x}_{10} \lor \bar{x}_6\bar{x}_{10}\bar{x}_{12} \lor x_8\bar{x}_{13}\bar{x}_{15} \lor \bar{x}_8x_{10}\bar{x}_{12}. \qquad (27)$$

This formula was discovered by constructing clauses in $2MN$ variables $p_{i,j}$ and $q_{i,j}$ for $1 \le i \le M$ and $1 \le j \le N$, where M is the maximum number of terms allowed in the DNF (here $M = 4$) and where

$$p_{i,j} \;=\; [\text{term } i \text{ contains } x_j], \qquad q_{i,j} \;=\; [\text{term } i \text{ contains } \bar{x}_j]. \qquad (28)$$

If the function is constrained to equal 1 at P specified points, we also use auxiliary variables $z_{i,k}$ for $1 \le i \le M$ and $1 \le k \le P$, one for each term at every such point.

Table 2 says that $f(1,1,0,0,\ldots,1) = 1$, and we can capture this specification by constructing the clause

$$(z_{1,1} \lor z_{2,1} \lor \cdots \lor z_{M,1}) \qquad (29)$$

together with the clauses

$$(\bar{z}_{i,1} \lor \bar{q}_{i,1}) \land (\bar{z}_{i,1} \lor \bar{q}_{i,2}) \land (\bar{z}_{i,1} \lor \bar{p}_{i,3}) \land (\bar{z}_{i,1} \lor \bar{p}_{i,4}) \land \cdots \land (\bar{z}_{i,1} \lor \bar{q}_{i,20}) \qquad (30)$$

for $1 \le i \le M$. Translation: (29) says that at least one of the terms in the DNF must evaluate to true; and (30) says that, if term i is true at the point $1100\ldots1$, it cannot contain $\bar{x}_1$ or $\bar{x}_2$ or x_3 or x_4 or $\cdots$ or $\bar{x}_{20}$.

Table 2 also tells us that $f(1,0,1,0,\ldots,1) = 0$. This specification corresponds to the clauses

$$(q_{i,1} \lor p_{i,2} \lor q_{i,3} \lor p_{i,4} \lor \cdots \lor q_{i,20}) \qquad (31)$$

for $1 \le i \le M$. (Each term of the DNF must be zero at the given point; thus either $\bar{x}_1$ or x_2 or $\bar{x}_3$ or x_4 or $\cdots$ or $\bar{x}_{20}$ must be present for each value of i.)

In general, every case where $f(x) = 1$ yields one clause like (29) of length M, plus MN clauses like (30) of length 2. Every case where $f(x) = 0$ yields M clauses like (31) of length N. We use $q_{i,j}$ when $x_j = 1$ at the point in question,

and $p_{i,j}$ when $x_j = 0$, for both (30) and (31). This construction is due to A. P. Kamath, N. K. Karmarkar, K. G. Ramakrishnan, and M. G. C. Resende [*Mathematical Programming* **57** (1992), 215–238], who presented many examples. From Table 2, with $M = 4$, $N = 20$, and $P = 16$, it generates 1360 clauses of total length 3904 in 224 variables; a SAT solver then finds a solution with $p_{1,1} = q_{1,1} = p_{1,2} = 0$, $q_{1,2} = 1$, ..., leading to (27).

The simplicity of (27) makes it plausible that the SAT solver has indeed psyched out the true nature of the hidden function $f(x)$. The chance of agreeing with the correct value 32 times out of 32 is only 1 in 2^{32}, so we seem to have overwhelming evidence in favor of that equation.

But no: Such reasoning is fallacious. The numbers in Table 2 actually arose in a completely different way, and Eq. (27) has essentially *no* credibility as a predictor of $f(x)$ for any other values of x! (See exercise 53.) The fallacy comes from the fact that short-DNF Boolean functions of 20 variables are not at all rare; there are many more than 2^{32} of them.

On the other hand, when we *do* know that the hidden function $f(x)$ has a DNF with at most M terms (although we know nothing else about it), the clauses (29)–(31) give us a nice way to discover those terms, provided that we also have a sufficiently large and unbiased "training set" of observed values.

For example, let's assume that (27) actually *is* the function in the box. If we examine $f(x)$ at 32 random points x, we don't have enough data to make any deductions. But 100 random training points will almost always home in on the correct solution (27). This calculation typically involves 3942 clauses in 344 variables; yet it goes quickly, needing only about 100 million accesses to memory.

One of the author's experiments with a 100-element training set yielded

$$\hat{f}(x_1, \ldots, x_{20}) = \bar{x}_2\bar{x}_3\bar{x}_{10} \vee x_3\bar{x}_6\bar{x}_{10}\bar{x}_{12} \vee x_8\bar{x}_{13}\bar{x}_{15} \vee \bar{x}_8 x_{10}\bar{x}_{12}, \qquad (32)$$

which is close to the truth but not quite exact. (Exercise 59 proves that $\hat{f}(x)$ is equal to $f(x)$ more than 97% of the time.) Further study of this example showed that another nine training points were enough to deduce $f(x)$ uniquely, thus obtaining 100% confidence (see exercise 61).

Bounded model checking. Some of the most important applications of SAT solvers in practice are related to the verification of hardware or software, because designers generally want some kind of assurance that particular implementations correctly meet their specifications.

A typical design can usually be modeled as a *transition relation* between Boolean vectors $X = x_1 \ldots x_n$ that represent the possible states of a system. We write $X \to X'$ if state X at time t can be followed by state X' at time $t + 1$. The task in general is to study sequences of state transitions

$$X_0 \to X_1 \to X_2 \to \cdots \to X_r, \qquad (33)$$

and to decide whether or not there are sequences that have special properties. For example, we hope that there's no such sequence for which X_0 is an "initial state" and X_r is an "error state"; otherwise there'd be a bug in the design.

Kamath
Karmarkar
Ramakrishnan
Resende
fallacious
training set
author
bounded model checking–
verification
model checking+
transition relation
dynamical system, discrete
bug

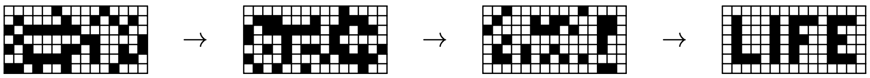


Fig. 35. Conway's rule (35) defines these three successive transitions.

auxiliary variables
Tseytin encodings
model
Conway
Life
bitmaps
cellular automaton
no-player game
universal
Berlekamp
Guy
BDD+

Questions like this are readily expressed as satisfiability problems: Each state X_t is a vector of Boolean variables $x_{t1} \ldots x_{tn}$, and each transition relation can be represented by a set of m clauses $T(X_t, X_{t+1})$ that must be satisfied. These clauses $T(X, X')$ involve $2n$ variables $\{x_1, \ldots, x_n, x'_1, \ldots, x'_n\}$, together with q auxiliary variables $\{y_1, \ldots, y_q\}$ that might be needed to express Boolean formulas in clause form as we did with the Tseytin encodings in (24). Then the existence of sequence (33) is equivalent to the satisfiability of mr clauses

$$T(X_0, X_1) \wedge T(X_1, X_2) \wedge \cdots \wedge T(X_{r-1}, X_r) \qquad (34)$$

in the $n(r+1)+qr$ variables $\{x_{tj} \mid 0 \le t \le r,\ 1 \le j \le n\} \cup \{y_{tk} \mid 0 \le t < r,\ 1 \le k \le q\}$. We've essentially "unrolled" the sequence (33) into r copies of the transition relation, using variables x_{tj} for state X_t and y_{tk} for the auxiliary quantities in $T(X_t, X_{t+1})$. Additional clauses can now be added to specify constraints on the initial state X_0 and/or the final state X_r, as well as any other conditions that we want to impose on the sequence.

This general setup is called "bounded model checking," because we're using it to check properties of a model (a transition relation), and because we're considering only sequences that have a bounded number of transitions, r.

John Conway's fascinating *Game of Life* provides a particularly instructive set of examples that illustrate basic principles of bounded model checking. The states X of this game are two-dimensional bitmaps, corresponding to arrays of square cells that are either alive (1) or dead (0). Every bitmap X has a unique successor X', determined by the action of a simple 3×3 cellular automaton: Suppose cell x has the eight neighbors $\{x_{\rm NW}, x_{\rm N}, x_{\rm NE}, x_{\rm W}, x_{\rm E}, x_{\rm SW}, x_{\rm S}, x_{\rm SE}\}$, and let $\nu = x_{\rm NW} + x_{\rm N} + x_{\rm NE} + x_{\rm W} + x_{\rm E} + x_{\rm SW} + x_{\rm S} + x_{\rm SE}$ be the number of neighbors that are alive at time t. Then x is alive at time $t+1$ if and only if either (a) $\nu = 3$, or (b) $\nu = 2$ and x is alive at time t. Equivalently, the transition rule

$$x' \;=\; \bigl[\,2 < x_{\rm NW} + x_{\rm N} + x_{\rm NE} + x_{\rm W} + \tfrac{1}{2}x + x_{\rm E} + x_{\rm SW} + x_{\rm S} + x_{\rm SE} < 4\,\bigr] \qquad (35)$$

holds at every cell x. (See, for example, Fig. 35, where the live cells are black.)

Conway called Life a "no-player game," because it involves no strategy: Once an initial state X_0 has been set up, all subsequent states X_1, X_2, ... are completely determined. Yet, in spite of the simple rules, he also proved that Life is inherently complicated and unpredictable, indeed beyond human comprehension, in the sense that it is universal: *Every finite, discrete, deterministic system, however complex, can be simulated faithfully by some finite initial state X_0 of Life.* [See Berlekamp, Conway, and Guy, *Winning Ways* (2004), Chapter 25.]

In exercises 7.1.4–160 through 162, we've already seen some of the amazing Life histories that are possible, using BDD methods. And many further aspects of Life can be explored with SAT methods, because SAT solvers can often deal

with many more variables. For example, Fig. 35 was discovered by using $7 \times 15 = 105$ variables for each state X_0, X_1, X_2, X_3. The values of X_3 were obviously predetermined; but the other $105 \times 3 = 315$ variables had to be computed, and BDDs can't handle that many. Moreover, additional variables were introduced to ensure that the initial state X_0 would have as few live cells as possible.

encoded
chessboard
grid
mobile

Here's the story behind Fig. 35, in more detail: Since Life is two-dimensional, we use variables x_{ij} instead of x_j to indicate the states of individual cells, and x_{tij} instead of x_{tj} to indicate the states of cells at time t. We generally assume that $x_{tij} = 0$ for all cells outside of a given finite region, although the transition rule (35) can allow cells that are arbitrarily far away to become alive as Life goes on. In Fig. 35 the region was specified to be a 7×15 rectangle at each unit of time. Furthermore, configurations with three consecutive live cells on a boundary edge were forbidden, so that cells "outside the box" wouldn't be activated.

The transitions $T(X_t, X_{t+1})$ can be encoded without introducing additional variables, but only if we introduce 190 rather long clauses for each cell not on the boundary. There's a better way, based on the binary tree approach underlying (20) and (21) above, which requires only about 63 clauses of size ≤ 3, together with about 14 auxiliary variables per cell. This approach (see exercise 65) takes advantage of the fact that many intermediate calculations can be shared. For example, cells x and x_{W} have four neighbors $\{x_{\text{NW}}, x_{\text{N}}, x_{\text{SW}}, x_{\text{S}}\}$ in common; so we need to compute $x_{\text{NW}} + x_{\text{N}} + x_{\text{SW}} + x_{\text{S}}$ only once, not twice.

The clauses that correspond to a four-step sequence $X_0 \to X_1 \to X_2 \to X_3 \to X_4$ leading to $X_4 =$ **LIFE** turn out to be unsatisfiable without going outside of the 7×15 frame. (Only 10 gigamems of calculation were needed to establish this fact, using Algorithm C below, even though roughly 34000 clauses in 9000 variables needed to be examined!) So the next step in the preparation of Fig. 35 was to try $X_3 =$ **LIFE**; and this trial succeeded. Additional clauses, which permitted X_0 to have at most 39 live cells, led to the solution shown, at a cost of about 17 gigamems; and that solution is optimum, because a further run (costing 12 gigamems) proved that there's no solution with at most 38.

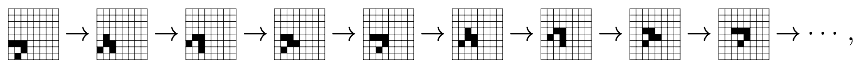
Let's look for a moment at some of the patterns that can occur on a chessboard, an 8×8 grid. Human beings will never be able to contemplate more than a tiny fraction of the 2^{64} states that are possible; so we can be fairly sure that "Lifenthusiasts" haven't already explored every tantalizing configuration that exists, even on such a small playing field.

One nice way to look for a sequence of interesting Life transitions is to assert that no cell stays alive more than four steps in a row. Let us therefore say that a *mobile* Life path is a sequence of transitions $X_0 \to X_1 \to \cdots \to X_r$ with the additional property that we have

$$(\bar{x}_{tij} \vee \bar{x}_{(t+1)ij} \vee \bar{x}_{(t+2)ij} \vee \bar{x}_{(t+3)ij} \vee \bar{x}_{(t+4)ij}), \qquad \text{for } 0 \leq t \leq r-4. \tag{36}$$

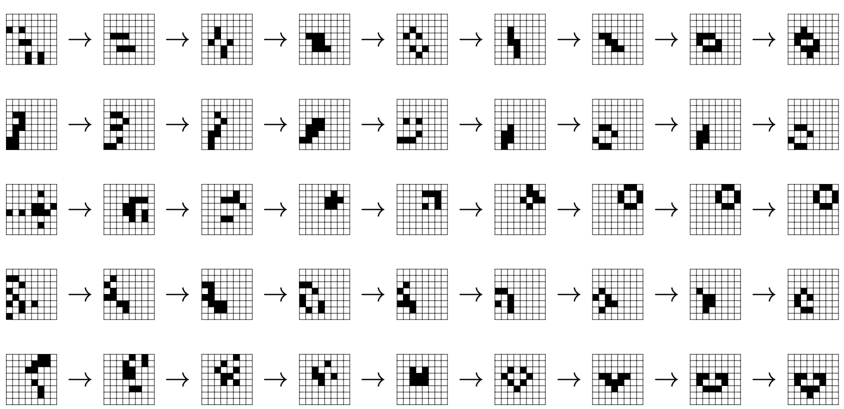
To avoid trivial solutions we also insist that X_r is not entirely dead. For example, if we impose rule (36) on a chessboard, with x_{tij} permitted to be alive only if $1 \leq i, j \leq 8$, and with the further condition that at most five cells are alive in each

generation, a SAT solver can quickly discover interesting mobile paths such as

 (37)

which last quite awhile before leaving the board. And indeed, the five-celled object that moves so gracefully in this path is R. K. Guy's famous *glider* (1970), which is surely the most interesting small creature in Life's universe. The glider moves diagonally, recreating a shifted copy of itself after every four steps.

Interesting mobile paths appear also if we restrict the population at each time to $\{6,7,8,9,10\}$ instead of $\{1,2,3,4,5\}$. For example, here are some of the first such paths that the author's solver came up with, having length $r = 8$:



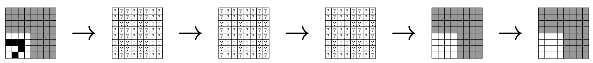
These paths illustrate the fact that symmetry can be gained, but never lost, as Life evolves deterministically. Marvelous designs are spawned in the process. In each of these sequences the next bitmap, X_9, would break our ground rules: The population immediately after X_8 grows to 12 in the first and last examples, but shrinks to 5 in the second-from-last; and the path becomes immobile in the other two. Indeed, we have $X_5 = X_7$ in the second example, hence $X_6 = X_8$ and $X_7 = X_9$, etc. Such a repeating pattern is called an *oscillator* of period 2. The third example ends with an oscillator of period 1, known as a "still life."

What are the ultimate destinations of these paths? The first one becomes still, with $X_{69} = X_{70}$; and the fourth becomes *very* still, with $X_{12} = 0$! The fifth is the most fascinating of the group, because it continues to produce ever more elaborate valentine shapes, then proceeds to dance and sparkle, until finally beginning to twinkle with period 2 starting at time 177. Thus its members X_2 through X_7 qualify as "Methuselahs," defined by Martin Gardner as "Life patterns of population less than 10 that do not become stable within 50 generations." (A predictable pattern, like the glider or an oscillator, is called *stable*.)

SAT solvers are basically useless for the study of Methuselahs, because the state space becomes too large. But they are quite helpful when we want to illuminate many other aspects of Life, and exercises 66–85 discuss some notable instances. We will consider one more instructive example before moving on,

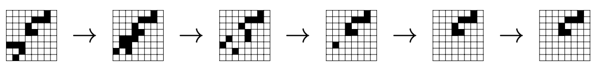
Guy
glider
population
Knuth
symmetry
oscillator
cyclic patterns
still life
Methuselahs
Gardner
stable

namely an application to "eaters." Consider a Life path of the form

X_0 =  = X_5, (38)

where the gray cells form a still life and the cells of X_1, X_2, X_3 are unknown. Thus $X_4 = X_5$ and $X_0 = X_5 +$ glider. Furthermore we require that the still life X_5 does not interact with the glider's parent, ; see exercise 77. The idea is that a glider will be gobbled up if it happens to glide into this particular still life, and the still life will rapidly reconstitute itself as if nothing had happened.

Algorithm C almost instantaneously (well, after about 100 megamems) finds

 (39)

the four-step eater first observed in action by R. W. Gosper in 1971.

Applications to mutual exclusion. Let's look now at how bounded model checking can help us to prove that algorithms are correct. (Or incorrect.) Some of the most challenging issues of verification arise when we consider parallel processes that need to synchronize their concurrent behavior. To simplify our discussion it will be convenient to tell a little story about Alice and Bob.

Alice and Bob are casual friends who share an apartment. One of their joint rooms is special: When they're in that critical room, which has two doors, they don't want the other person to be present. Furthermore, being busy people, they don't want to interrupt each other needlessly. So they agree to control access to the room by using an indicator light, which can be switched on or off.

The first protocol they tried can be characterized by symmetrical algorithms:

A0. Maybe go to A1.	B0. Maybe go to B1.	
A1. If l go to A1, else to A2.	B1. If l go to B1, else to B2.	
A2. Set $l \leftarrow 1$, go to A3.	B2. Set $l \leftarrow 1$, go to B3.	(40)
A3. Critical, go to A4.	B3. Critical, go to B4.	
A4. Set $l \leftarrow 0$, go to A0.	B4. Set $l \leftarrow 0$, go to B0.	

At any instant of time, Alice is in one of five states, $\{A0, A1, A2, A3, A4\}$, and the rules of her program show how that state might change. In state A0 she isn't interested in the critical room; but she goes to A1 when she does wish to use it. She reaches that objective in state A3. Similar remarks apply to Bob. When the indicator light is on ($l = 1$), they wait until the other person has exited the room and switched the light back off ($l = 0$).

Alice and Bob don't necessarily operate at the same speed. But they're allowed to dawdle only when in the "maybe" state A0 or B0. More precisely, we model the situation by converting every relevant scenario into a discrete sequence of state transitions. At every time $t = 0, 1, 2, \ldots$, either Alice or Bob (but not both) will perform the command associated with their current state, thereby perhaps changing to a different state at time $t + 1$. This choice is nondeterministic.

Only four kinds of primitive commands are permitted in the procedures we shall study, all of which are illustrated in (40): (1) "Maybe go to s"; (2) "Critical,

eaters
Gosper
parallel processes
Alice–
Bob–
"maybe" state
nondeterministic

exclusion clauses
bumped
initial state
critical section

go to s"; (3) "Set $v \leftarrow b$, go to s"; and (4) "If v go to s_1, else to s_0". Here s denotes a state name, v denotes a shared Boolean variable, and b is 0 or 1.

Unfortunately, Alice and Bob soon learned that protocol (40) is unreliable: One day she went from A1 to A2 and he went from B1 to B2, before either of them had switched the indicator on. Embarrassment (A3 and B3) followed.

They could have discovered this problem in advance, if they'd converted the state transitions of (40) into clauses for bounded model checking, as in (33), then applied a SAT solver. In this case the vector X_t that corresponds to time t consists of Boolean variables that encode each of their current states, as well as the current value of l. We can, for example, have eleven variables $\mathrm{A0}_t, \mathrm{A1}_t, \mathrm{A2}_t, \mathrm{A3}_t, \mathrm{A4}_t, \mathrm{B0}_t, \mathrm{B1}_t, \mathrm{B2}_t, \mathrm{B3}_t, \mathrm{B4}_t, l_t$, together with ten binary exclusion clauses $(\overline{\mathrm{A0}_t} \vee \overline{\mathrm{A1}_t})$, $(\overline{\mathrm{A0}_t} \vee \overline{\mathrm{A2}_t})$, $\ldots$, $(\overline{\mathrm{A3}_t} \vee \overline{\mathrm{A4}_t})$ to ensure that Alice is in at most one state, and with ten similar clauses for Bob. There's also a variable $@_t$, which is true or false depending on whether Alice or Bob executes their program step at time t. (We say that Alice was "bumped" if $@_t = 1$, and Bob was bumped if $@_t = 0$.)

If we start with the initial state X_0 defined by unit clauses

$$\mathrm{A0}_0 \wedge \overline{\mathrm{A1}_0} \wedge \overline{\mathrm{A2}_0} \wedge \overline{\mathrm{A3}_0} \wedge \overline{\mathrm{A4}_0} \wedge \mathrm{B0}_0 \wedge \overline{\mathrm{B1}_0} \wedge \overline{\mathrm{B2}_0} \wedge \overline{\mathrm{B3}_0} \wedge \overline{\mathrm{B4}_0} \wedge \bar{l}_0, \qquad (41)$$

the following clauses for $0 \le t < r$ (discussed in exercise 87) will emulate the first r steps of every legitimate scenario defined by (40):

$$\begin{array}{lll}
(@_t \vee \overline{\mathrm{A0}_t} \vee \mathrm{A0}_{t+1}) & (\overline{@_t} \vee \overline{\mathrm{A0}_t} \vee \mathrm{A0}_{t+1} \vee \mathrm{A1}_{t+1}) & (@_t \vee \overline{\mathrm{B0}_t} \vee \mathrm{B0}_{t+1} \vee \mathrm{B1}_{t+1}) \\
(@_t \vee \overline{\mathrm{A1}_t} \vee \mathrm{A1}_{t+1}) & (\overline{@_t} \vee \overline{\mathrm{A1}_t} \vee \bar{l}_t \vee \mathrm{A1}_{t+1}) & (@_t \vee \overline{\mathrm{B1}_t} \vee \bar{l}_t \vee \mathrm{B1}_{t+1}) \\
(@_t \vee \overline{\mathrm{A2}_t} \vee \mathrm{A2}_{t+1}) & (\overline{@_t} \vee \overline{\mathrm{A1}_t} \vee l_t \vee \mathrm{A2}_{t+1}) & (@_t \vee \overline{\mathrm{B1}_t} \vee l_t \vee \mathrm{B2}_{t+1}) \\
(@_t \vee \overline{\mathrm{A3}_t} \vee \mathrm{A3}_{t+1}) & (\overline{@_t} \vee \overline{\mathrm{A2}_t} \vee \mathrm{A3}_{t+1}) & (@_t \vee \overline{\mathrm{B2}_t} \vee \mathrm{B3}_{t+1}) \\
(@_t \vee \overline{\mathrm{A4}_t} \vee \mathrm{A4}_{t+1}) & (\overline{@_t} \vee \overline{\mathrm{A2}_t} \vee l_{t+1}) & (@_t \vee \overline{\mathrm{B2}_t} \vee l_{t+1}) \\
(\overline{@_t} \vee \overline{\mathrm{B0}_t} \vee \mathrm{B0}_{t+1}) & (\overline{@_t} \vee \overline{\mathrm{A3}_t} \vee \mathrm{A4}_{t+1}) & (@_t \vee \overline{\mathrm{B3}_t} \vee \mathrm{B4}_{t+1}) \\
(\overline{@_t} \vee \overline{\mathrm{B1}_t} \vee \mathrm{B1}_{t+1}) & (\overline{@_t} \vee \overline{\mathrm{A4}_t} \vee \mathrm{A0}_{t+1}) & (@_t \vee \overline{\mathrm{B4}_t} \vee \mathrm{B0}_{t+1}) \\
(\overline{@_t} \vee \overline{\mathrm{B2}_t} \vee \mathrm{B2}_{t+1}) & (\overline{@_t} \vee \overline{\mathrm{A4}_t} \vee \bar{l}_{t+1}) & (@_t \vee \overline{\mathrm{B4}_t} \vee \bar{l}_{t+1}) \\
(\overline{@_t} \vee \overline{\mathrm{B3}_t} \vee \mathrm{B3}_{t+1}) & (\overline{@_t} \vee l_t \vee \mathrm{A2}_t \vee \mathrm{A4}_t \vee \bar{l}_{t+1}) & (@_t \vee l_t \vee \mathrm{B2}_t \vee \mathrm{B4}_t \vee \bar{l}_{t+1}) \\
(\overline{@_t} \vee \overline{\mathrm{B4}_t} \vee \mathrm{B4}_{t+1}) & (\overline{@_t} \vee \bar{l}_t \vee \mathrm{A2}_t \vee \mathrm{A4}_t \vee l_{t+1}) & (@_t \vee \bar{l}_t \vee \mathrm{B2}_t \vee \mathrm{B4}_t \vee l_{t+1})
\end{array} \qquad (42)$$

If we now add the unit clauses $(\mathrm{A3}_r)$ and $(\mathrm{B3}_r)$, the resulting set of $13 + 50r$ clauses in $11+12r$ variables is readily satisfiable when $r = 6$, thereby proving that the critical room might indeed be jointly occupied. (Incidentally, standard terminology for mutual exclusion protocols would say that "two threads concurrently execute a *critical section*"; but we shall continue with our roommate metaphor.)

Back at the drawing board, one idea is to modify (40) by letting Alice use the room only when $l = 1$, but letting Bob in when $l = 0$:

A0. Maybe go to A1.	B0. Maybe go to B1.	
A1. If l go to A2, else to A1.	B1. If l go to B1, else to B2.	(43)
A2. Critical, go to A3.	B2. Critical, go to B3.	
A3. Set $l \leftarrow 0$, go to A0.	B3. Set $l \leftarrow 1$, go to B0.	

Computer tests with $r = 100$ show that the corresponding clauses are unsatisfiable; thus mutual exclusion is apparently guaranteed by (43).

deadlock
reboot
Dijkstra
starvation

But (43) is a nonstarter, because it imposes an intolerable cost: Alice can't use the room k times until Bob has already done so! Scrap that.

How about installing another light, so that each person controls one of them?

A0. Maybe go to A1.	B0. Maybe go to B1.	
A1. If b go to A1, else to A2.	B1. If a go to B1, else to B2.	
A2. Set $a \leftarrow 1$, go to A3.	B2. Set $b \leftarrow 1$, go to B3.	(44)
A3. Critical, go to A4.	B3. Critical, go to B4.	
A4. Set $a \leftarrow 0$, go to A0.	B4. Set $b \leftarrow 0$, go to B0.	

No; this suffers from the same defect as (40). But maybe we can cleverly switch the order of steps 1 and 2:

A0. Maybe go to A1.	B0. Maybe go to B1.	
A1. Set $a \leftarrow 1$, go to A2.	B1. Set $b \leftarrow 1$, go to B2.	
A2. If b go to A2, else to A3.	B2. If a go to B2, else to B3.	(45)
A3. Critical, go to A4.	B3. Critical, go to B4.	
A4. Set $a \leftarrow 0$, go to A0.	B4. Set $b \leftarrow 0$, go to B0.	

Yes! Exercise 95 proves easily that this protocol does achieve mutual exclusion.

Alas, however, a new problem now arises, namely the problem known as "deadlock" or "livelock." Alice and Bob can get into states A2 and B2, after which they're stuck — each waiting for the other to go critical.

In such cases they could agree to "reboot" somehow. But that would be a cop-out; they really seek a better solution. And they aren't alone: Many people have struggled with this surprisingly delicate problem over the years, and several solutions (both good and bad) appear in the exercises below. Edsger Dijkstra, in some pioneering lecture notes entitled *Cooperating Sequential Processes* [Technological University Eindhoven (September 1965), §2.1], thought of an instructive way to improve on (45):

A0. Maybe go to A1.	B0. Maybe go to B1.	
A1. Set $a \leftarrow 1$, go to A2.	B1. Set $b \leftarrow 1$, go to B2.	
A2. If b go to A3, else to A4.	B2. If a go to B3, else to B4.	(46)
A3. Set $a \leftarrow 0$, go to A1.	B3. Set $b \leftarrow 0$, go to B1.	
A4. Critical, go to A5.	B4. Critical, go to B5.	
A5. Set $a \leftarrow 0$, go to A0.	B5. Set $b \leftarrow 0$, go to B0.	

But he realized that this too is unsatisfactory, because it permits scenarios in which Alice, say, might wait forever while Bob repeatedly uses the critical room. (Indeed, if Alice and Bob are in states A1 and B2, she might go to A2, A3, then A1, thereby letting him run to B4, B5, B0, B1, and B2; they're back where they started, yet she's made no progress.)

The existence of this problem, called *starvation*, can also be detected via bounded model checking. The basic idea (see exercise 91) is that starvation occurs if and only if there is a loop of transitions

$$X_0 \to X_1 \to \cdots \to X_p \to X_{p+1} \to \cdots \to X_r = X_p \qquad (47)$$

such that (i) Alice and Bob each are bumped at least once during the loop; and (ii) at least one of them is never in a "maybe" or "critical" state during the loop.

Peterson
simple path
longest simple path
invariants

And those conditions are easily encoded into clauses, because we can identify the variables for time r with the variables for time p, and we can append the clauses

$$(\overline{@_p} \vee \overline{@_{p+1}} \vee \cdots \vee \overline{@_{r-1}}) \ \wedge\ (@_p \vee @_{p+1} \vee \cdots \vee @_{r-1}) \tag{48}$$

to guarantee (i). Condition (ii) is simply a matter of appending unit clauses; for example, to test whether Alice can be starved by (46), the relevant clauses are $\overline{\mathrm{A0}_p} \wedge \overline{\mathrm{A0}_{p+1}} \wedge \cdots \wedge \overline{\mathrm{A0}_{r-1}} \wedge \overline{\mathrm{A4}_p} \wedge \overline{\mathrm{A4}_{p+1}} \wedge \cdots \wedge \overline{\mathrm{A4}_{r-1}}$.

The deficiencies of (43), (45), and (46) can all be viewed as instances of starvation, because (47) and (48) are satisfiable (see exercise 90). Thus we can use bounded model checking to find counterexamples to *any* unsatisfactory protocol for mutual exclusion, either by exhibiting a scenario in which Alice and Bob are both in the critical room or by exhibiting a feasible starvation cycle (47).

Of course we'd like to go the other way, too: If a protocol has no counterexamples for, say, $r = 100$, we still might not know that it is really reliable; a counterexample might exist only when r is extremely large. Fortunately there are ways to obtain decent upper bounds on r, so that bounded model checking can be used to prove correctness as well as to demonstrate incorrectness. For example, we can verify the simplest known correct solution to Alice and Bob's problem, a protocol by G. L. Peterson [*Information Proc. Letters* **12** (1981), 115–116], who noticed that a careful combination of (43) and (45) actually suffices:

A0. Maybe go to A1.	B0. Maybe go to B1.	
A1. Set $a \leftarrow 1$, go to A2.	B1. Set $b \leftarrow 1$, go to B2.	
A2. Set $l \leftarrow 0$, go to A3.	B2. Set $l \leftarrow 1$, go to B3.	
A3. If b go to A4, else to A5.	B3. If a go to B4, else to B5.	(49)
A4. If l go to A5, else to A3.	B4. If l go to B3, else to B5.	
A5. Critical, go to A6.	B5. Critical, go to B6.	
A6. Set $a \leftarrow 0$, go to A0.	B6. Set $b \leftarrow 0$, go to B0.	

Now there are *three* signal lights, a, b, and l—one controlled by Alice, one controlled by Bob, and one switchable by both.

To show that states A5 and B5 can't be concurrent, we can observe that the shortest counterexample will not repeat any state twice; in other words, it will be a *simple* path of transitions (33). Thus we can assume that r is at most the total number of states. However, (49) has $7 \times 7 \times 2 \times 2 \times 2 = 392$ states; that's a finite bound, not really out of reach for a good SAT solver on this particular problem, but we can do much better. For example, it's not hard to devise clauses that are satisfiable if and only if there's a simple path of length $\le r$ (see exercise 92), and in this particular case the longest simple path turns out to have only 54 steps.

We can in fact do better yet by using the important notion of *invariants*, which we encountered in Section 1.2.1 and have seen repeatedly throughout this series of books. Invariant assertions are the key to most proofs of correctness, so it's not surprising that they also give a significant boost to bounded model checking. Formally speaking, if $\Phi(X)$ is a Boolean function of the state vector X, we say that Φ is invariant if $\Phi(X)$ implies $\Phi(X')$ whenever $X \to X'$. For example,

it's not hard to see that the following clauses are invariant with respect to (49):

$$\Phi(X) \;=\; (\mathrm{A0}\vee\mathrm{A1}\vee\mathrm{A2}\vee\mathrm{A3}\vee\mathrm{A4}\vee\mathrm{A5}\vee\mathrm{A6}) \wedge (\mathrm{B0}\vee\mathrm{B1}\vee\mathrm{B2}\vee\mathrm{B3}\vee\mathrm{B4}\vee\mathrm{B5}\vee\mathrm{B6})$$
$$\wedge\,(\overline{\mathrm{A0}}\vee\bar a)\wedge(\overline{\mathrm{A1}}\vee\bar a)\wedge(\overline{\mathrm{A2}}\vee a)\wedge(\overline{\mathrm{A3}}\vee a)\wedge(\overline{\mathrm{A4}}\vee a)\wedge(\overline{\mathrm{A5}}\vee a)\wedge(\overline{\mathrm{A6}}\vee a)$$
$$\wedge\,(\overline{\mathrm{B0}}\vee\bar b)\wedge(\overline{\mathrm{B1}}\vee\bar b)\wedge(\overline{\mathrm{B2}}\vee b)\wedge(\overline{\mathrm{B3}}\vee b)\wedge(\overline{\mathrm{B4}}\vee b)\wedge(\overline{\mathrm{B5}}\vee b)\wedge(\overline{\mathrm{B6}}\vee b). \qquad (50)$$

(The clause $\overline{\mathrm{A0}}\vee\bar a$ says that $a=0$ when Alice is in state A0, etc.) And we can use a SAT solver to *prove* that Φ is invariant, by showing that the clauses

$$\Phi(X)\;\wedge\;(X\to X')\;\wedge\;\neg\Phi(X') \qquad (51)$$

are *unsatisfiable*. Furthermore $\Phi(X_0)$ holds for the initial state X_0, because $\neg\Phi(X_0)$ is unsatisfiable. (See exercise 93.) Therefore $\Phi(X_t)$ is true for all $t\ge 0$, by induction, and we may add these helpful clauses to all of our formulas.

The invariant (50) reduces the total number of states by a factor of 4. And the real clincher is the fact that the clauses

$$(X_0\to X_1\to\cdots\to X_r)\wedge\Phi(X_0)\wedge\Phi(X_1)\wedge\cdots\wedge\Phi(X_r)\wedge \mathrm{A5}_r\wedge \mathrm{B5}_r, \qquad (52)$$

where X_0 is *not* required to be the initial state, turn out to be unsatisfiable when $r=3$. In other words, there's no way to go back more than two steps from a bad state, without violating the invariant. We can conclude that mutual exclusion needs to be verified for (49) only by considering paths of length 2(!). Furthermore, similar ideas (exercise 98) show that (49) is starvation-free.

Caveat: Although (49) is a correct protocol for mutual exclusion according to Alice and Bob's ground rules, it *cannot* be used safely on most modern computers unless special care is taken to synchronize cache memories and write buffers. The reason is that hardware designers use all sorts of trickery to gain speed, and those tricks might allow one process to see $a=0$ at time $t+1$ even though another process has set $a\leftarrow 1$ at time t. We have developed the algorithms above by assuming a model of parallel computation that Leslie Lamport has called *sequential consistency* [*IEEE Trans.* **C-28** (1979), 690–691].

Digital tomography. Another set of appealing questions amenable to SAT solving comes from the study of binary images for which partial information is given. Consider, for example, Fig. 36, which shows the "Cheshire cat" of Section 7.1.3 in a new light. This image is an $m\times n$ array of Boolean variables $(x_{i,j})$, with $m=25$ rows and $n=30$ columns: The upper left corner element, $x_{1,1}$, is 0, representing white; and $x_{1,24}=1$ corresponds to the lone black pixel in the top row. We are given the row sums $r_i=\sum_{j=1}^{n}x_{i,j}$ for $1\le i\le m$ and the column sums $c_j=\sum_{i=1}^{m}x_{i,j}$ for $1\le j\le n$, as well as both sets of sums in the 45° diagonal directions, namely

$$a_d=\sum_{i+j=d+1}x_{i,j} \quad\text{and}\quad b_d=\sum_{i-j=d-n}x_{i,j} \quad\text{for } 0<d<m+n. \qquad (53)$$

To what extent can such an image be reconstructed from its sums r_i, c_j, a_d, and b_d? Small examples are often uniquely determined by these Xray-like projections (see exercise 103). But the discrete nature of pixel images makes the reconstruction problem considerably more difficult than the corresponding

initial state
induction
cache memories
write buffers
parallel computation
Lamport
sequential consistency
tomcat
Cheshire cat
diagonal
Xray
pixel images

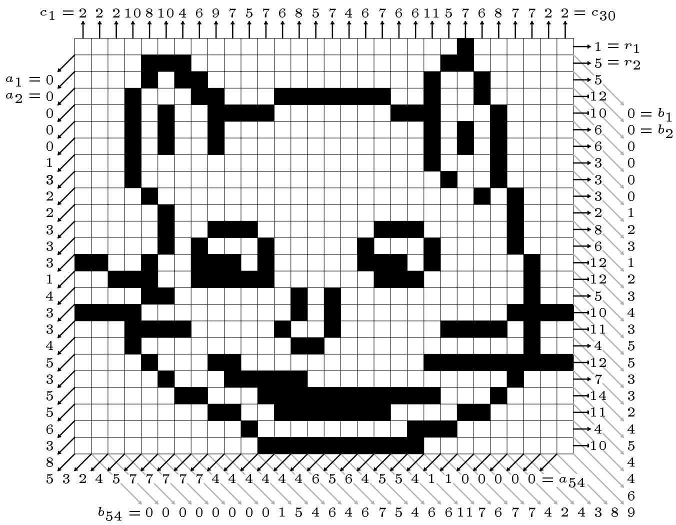
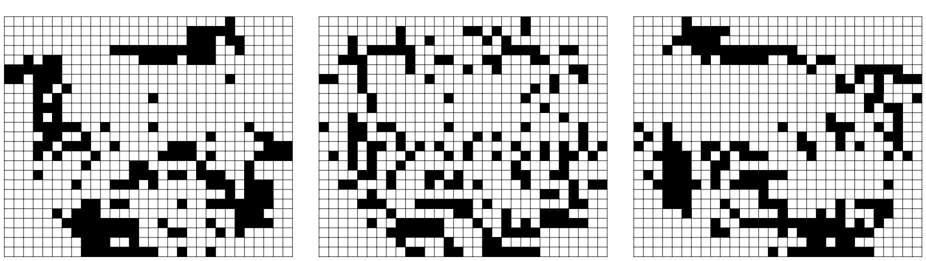


Fig. 36. An array of black and white pixels together with its row sums r_i, column sums c_j, and diagonal sums a_d, b_d.

continuous problem, in which projections from many different angles are available. Notice, for example, that the classical "8 queens problem" — to place eight nonattacking queens on a chessboard — is equivalent to solving an 8×8 digital tomography problem with the constraints $r_i = 1$, $c_j = 1$, $a_d \leq 1$, and $b_d \leq 1$.

The constraints of Fig. 36 appear to be quite strict, so we might expect that most of the pixels $x_{i,j}$ are determined uniquely by the given sums. For instance, the fact that $a_1 = \cdots = a_5 = 0$ tells us that $x_{i,j} = 0$ whenever $i + j \leq 6$; and similar deductions are possible at all four corners of the image. A crude "ballpark estimate" suggests that we're given a few more than 150 sums, most of which occupy 5 bits each; hence we have roughly $150 \times 5 = 750$ bits of data, from which we wish to reconstruct $25 \times 30 = 750$ pixels $x_{i,j}$. Actually, however, this problem turns out to have many billions of solutions (see Fig. 37), most of which aren't catlike! Exercise 106 provides a less crude estimate, which shows that this abundance of solutions isn't really surprising.

8 queens problem
queens
chessboard
lexicographic order



(a) lexicographically first; (b) maximally different; (c) lexicographically last.

Fig. 37. Extreme solutions to the constraints of Fig. 36.

A digital tomography problem such as Fig. 36 is readily represented as a sequence of clauses to be satisfied, because each of the individual requirements is just a special case of the cardinality constraints that we've already considered in the clauses of (18)–(21). This problem differs from the other instances of SAT that we've been discussing, primarily because it consists *entirely* of cardinality constraints: It is a question of solving $25 + 30 + 54 + 54 = 163$ simultaneous linear equations in 750 variables $x_{i,j}$, where each variable must be either 0 or 1. So it's essentially an instance of *integer programming* (IP), not an instance of satisfiability (SAT). On the other hand, Bailleux and Boufkhad devised clauses (20) and (21) precisely because they wanted to apply SAT solvers, not IP solvers, to digital tomography. In the case of Fig. 36, their method yields approximately 40,000 clauses in 9,000 variables, containing about 100,000 literals altogether.

Figure 37(b) illustrates a solution that differs as much as possible from Fig. 36. Thus it minimizes the sum $x_{1,24} + x_{2,5} + x_{2,6} + \cdots + x_{25,21}$ of the 182 variables that correspond to black pixels, over all 0-or-1-valued solutions to the linear equations. If we use linear programming to minimize that sum over $0 \le x_{i,j} \le 1$, *without* requiring the variables to be integers, we find almost instantly that the minimum value is ≈ 31.38 under these relaxed conditions; hence every black-and-white image must have at least 32 black pixels in common with Fig. 36. Furthermore, Fig. 37(b) — which can be computed in a few seconds by widely available IP solvers such as CPLEX — actually achieves this minimum. By contrast, state-of-the-art SAT solvers as of 2013 had great difficulty finding such an image, even when told that a 32-in-common solution is possible.

Parts (a) and (c) of Fig. 37 are, similarly, quite relevant to the current state of the SAT-solving art: They represent hundreds of individual SAT instances, where the first k variables are set to particular known values and we try to find a solution with the next variable either 0 or 1, respectively. Several of the subproblems that arose while computing rows 6 and 7 of Fig. 37(c) turned out to be quite challenging, although resolvable in a few hours; and similar problems, which correspond to different kinds of lexicographic order, apparently still lie beyond the reach of contemporary SAT-oriented methods. Yet IP solvers polish these problems off with ease. (See exercises 109 and 111.)

If we provide more information about an image, our chances of being able to reconstruct it uniquely are naturally enhanced. For example, suppose we also compute the numbers r'_i, c'_j, a'_d, and b'_d, which count the *runs of 1s* that occur in each row, column, and diagonal. (We have $r'_1 = 1$, $r'_2 = 2$, $r'_3 = 4$, and so on.) Given this additional data, we can show that Fig. 36 is the only solution, because a suitable set of clauses turns out to be unsatisfiable. Exercise 117 explains one way by which (20) and (21) can be modified so that they provide constraints based on the run counts. Furthermore, it isn't difficult to express even more detailed constraints, such as the assertion that "column 4 contains runs of respective lengths $(6, 1, 3)$," as a sequence of clauses; see exercise 438.

SAT examples — summary. We've now seen convincing evidence that simple Boolean clauses — ANDs of ORs of literals — are enormously versatile. Among

cardinality constraints
linear equations
integer programming
Bailleux
Boufkhad
linear programming relaxation
CPLEX
IP: Integer programming
lexicographic order
runs of 1s

other things, we've used them to encode problems of graph coloring, integer factorization, hardware fault testing, machine learning, model checking, and tomography. And indeed, Section 7.9 will demonstrate that 3SAT is the "poster child" for NP-complete problems in general: *Any* problem in NP — which is a huge class, essentially comprising all yes-or-no questions of size N whose affirmative answers are verifiable in $N^{O(1)}$ steps — can be formulated as an equivalent instance of 3SAT, without greatly increasing the problem size.

backtracking–
notation $F\,|\,l$
reduction of clauses
given literals
conditioning operation
commutative
notation $F\,|\,L$
recursive procedure

Backtracking for SAT. We've now seen a dizzying variety of intriguing and important examples of SAT that are begging to be solved. How shall we solve them?

Any instance of SAT that involves at least one variable can be solved systematically by choosing a variable and setting it to 0 or 1. Either of those choices gives us a smaller instance of SAT; so we can continue until reaching either an empty instance — which is trivially satisfiable, because no clauses need to be satisfied — or an instance that contains an empty clause. In the latter case we must back up and reconsider one of our earlier choices, proceeding in the same fashion until we either succeed or exhaust all the possibilities.

For example, consider again the formula F in (1). If we set $x_1 = 0$, F reduces to $\bar{x}_2 \wedge (x_2 \vee x_3)$, because the first clause $(x_1 \vee \bar{x}_2)$ loses its x_1, while the last two clauses contain $\bar{x}_1$ and are satisfied. It will be convenient to have a notation for this reduced problem; so let's write

$$F\,|\,\bar{x}_1 \;=\; \bar{x}_2 \wedge (x_2 \vee x_3). \tag{54}$$

Similarly, if we set $x_1 = 1$, we obtain the reduced problem

$$F\,|\,x_1 \;=\; (x_2 \vee x_3) \wedge \bar{x}_3 \wedge (\bar{x}_2 \vee x_3). \tag{55}$$

F is satisfiable if and only if we can satisfy either (54) or (55).

In general if F is any set of clauses and if l is any literal, then $F\,|\,l$ (read "F given l" or "F conditioned on l") is the set of clauses obtained from F by

- removing every clause that contains l; and
- removing $\bar{l}$ from every clause that contains $\bar{l}$.

This conditioning operation is commutative, in the sense that $F\,|\,l\,|\,l' = F\,|\,l'\,|\,l$ when $l' \neq \bar{l}$. If $L = \{l_1, \ldots, l_k\}$ is any set of strictly distinct literals, we can also write $F\,|\,L = F\,|\,l_1\,|\cdots|\,l_k$. In these terms, F is satisfiable if and only if $F\,|\,L = \emptyset$ for some such L, because the literals of L satisfy every clause of F when $F\,|\,L = \emptyset$.

The systematic strategy for SAT that was sketched above can therefore be formulated as the following recursive procedure $B(F)$, which returns the special value $\perp$ when F is unsatisfiable, otherwise it returns a set L that satisfies F:

$$B(F) = \begin{cases} \text{If } F = \emptyset, \text{ return } \emptyset. \ (F \text{ is trivially satisfiable.}) \\ \text{Otherwise if } \epsilon \in F, \text{ return } \perp. \ (F \text{ is unsatisfiable.}) \\ \text{Otherwise let } l \text{ be a literal in } F \text{ and set } L \leftarrow B(F\,|\,l). \\ \text{If } L \neq \perp, \text{ return } L \cup l. \text{ Otherwise set } L \leftarrow B(F\,|\,\bar{l}). \\ \text{If } L \neq \perp, \text{ return } L \cup \bar{l}. \text{ Otherwise return } \perp. \end{cases} \tag{56}$$

Let's try to flesh out this abstract algorithm by converting it to efficient code at a lower level. From our previous experience with backtracking, we know

that it will be crucial to have data structures that allow us to go quickly from F to $F \mid l$, then back again to F if necessary, when F is a set of clauses and l is a literal. In particular, we'll want a good way to find all of the clauses that contain a given literal.

A combination of sequential and linked structures suggests itself for this purpose, based on our experience with exact cover problems: We can represent each clause as a set of *cells*, where each cell p contains a literal $l = \texttt{L}(p)$ together with pointers $\texttt{F}(p)$ and $\texttt{B}(p)$ to other cells that contain l, in a doubly linked list. We'll also need $\texttt{C}(p)$, the number of the clause to which p belongs. The cells of clause C_i will be in consecutive locations $\texttt{START}(i) + j$, for $0 \le j < \texttt{SIZE}(i)$.

We will find it convenient to represent the literals x_k and $\bar{x}_k$, which involve variable x_k, by using the integers $2k$ and $2k+1$. With this convention we have

$$\bar{l} = l \oplus 1 \qquad \text{and} \qquad |l| = x_{l \gg 1}. \tag{57}$$

Our implementation of (56) will assume that the variables are $x_1, x_2, \ldots, x_n$; thus the $2n$ possible literals will be in the range $2 \le l \le 2n+1$.

Cells 0 through $2n+1$ are reserved for special purposes: Cell l is the head of the list for the occurrences of l in other cells. Furthermore $\texttt{C}(l)$ will be the length of that list, namely the number of currently active clauses in which l appears.

For example, the $m = 7$ ternary clauses R' of (7) might be represented internally in $2n + 2 + 3m = 31$ cells as follows, using these conventions:

$p =$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
$\texttt{L}(p) =$	–	–	–	–	–	–	–	–	–	–	9	7	3	8	7	5	6	5	3	8	4	3	8	6	2	9	6	4	7	4	2
$\texttt{F}(p) =$	–	–	30	21	29	17	26	28	22	25	9	7	3	8	11	5	6	15	12	13	4	18	19	16	2	10	23	20	14	27	24
$\texttt{B}(p) =$	–	–	24	12	20	15	16	11	13	10	25	14	18	19	28	17	23	5	21	22	27	3	8	26	30	9	6	29	7	4	2
$\texttt{C}(p) =$	–	–	2	3	3	2	3	3	3	2	7	7	7	6	6	6	5	5	5	4	4	4	3	3	3	2	2	2	1	1	1

The literals of each clause appear in decreasing order here; for example, the literals $\texttt{L}(p) = (8, 4, 3)$ in cells 19 through 21 represent the clause $x_4 \vee x_2 \vee \bar{x}_1$, which appears as the fourth clause, '$4\bar{1}2$' in (7). This ordering turns out to be quite useful, because we'll always choose the smallest unset variable as the l or $\bar{l}$ in (56); then l or $\bar{l}$ will always appear at the right of its clauses, and we can remove it or put it back by simply changing the relevant SIZE fields.

The clauses in this example have $\texttt{START}(i) = 31 - 3i$ for $1 \le i \le 7$, and $\texttt{SIZE}(i) = 3$ when computation begins.

Algorithm A (*Satisfiability by backtracking*). Given nonempty clauses $C_1 \wedge \cdots \wedge C_m$ on $n > 0$ Boolean variables $x_1 \ldots x_n$, represented as above, this algorithm finds a solution if and only if the clauses are satisfiable. It records its current progress in an array $m_1 \ldots m_n$ of "moves," whose significance is explained below.

A1. [Initialize.] Set $a \leftarrow m$ and $d \leftarrow 1$. (Here a represents the number of active clauses, and d represents the depth-plus-one in an implicit search tree.)

A2. [Choose.] Set $l \leftarrow 2d$. If $\texttt{C}(l) \le \texttt{C}(l+1)$, set $l \leftarrow l+1$. Then set $m_d \leftarrow (l \mathbin{\&} 1) + 4[\texttt{C}(l \oplus 1) = 0]$. (See below.) Terminate successfully if $\texttt{C}(l) = a$.

A3. [Remove $\bar{l}$.] Delete $\bar{l}$ from all active clauses; but go to A5 if that would make a clause empty. (We want to ignore $\bar{l}$, because we're making l true.)

data structures–
exact cover
cells
doubly linked
head of the list

move codes
pure literals

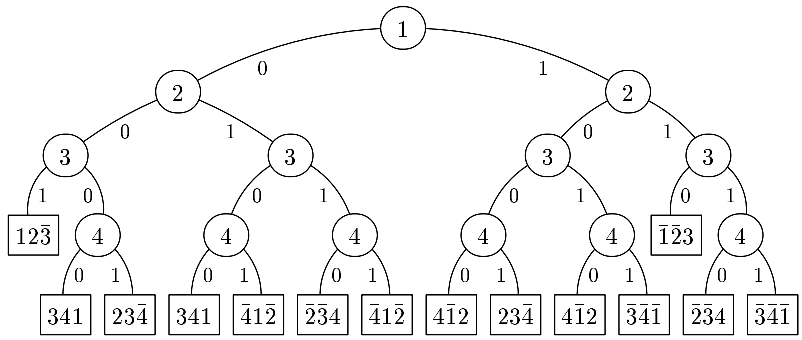


Fig. 38. The search tree that is implicitly traversed by Algorithm A, when that algorithm is applied to the eight unsatisfiable clauses R defined in (6). Branch nodes are labeled with the variable being tested; leaf nodes are labeled with a clause that is found to be contradicted.

A4. [Deactivate l's clauses.] Suppress all clauses that contain l. (Those clauses are now satisfied.) Then set $a \leftarrow a - \mathtt{C}(l)$, $d \leftarrow d + 1$, and return to A2.

A5. [Try again.] If $m_d < 2$, set $m_d \leftarrow 3 - m_d$, $l \leftarrow 2d + (m_d \,\&\, 1)$, and go to A3.

A6. [Backtrack.] Terminate unsuccessfully if $d = 1$ (the clauses are unsatisfiable). Otherwise set $d \leftarrow d - 1$ and $l \leftarrow 2d + (m_d \,\&\, 1)$.

A7. [Reactivate l's clauses.] Set $a \leftarrow a + \mathtt{C}(l)$, and unsuppress all clauses that contain l. (Those clauses are now unsatisfied, because l is no longer true.)

A8. [Unremove $\bar{l}$.] Reinstate $\bar{l}$ in all the active clauses that contain it. Then go back to A5. ▌

(See exercise 121 for details of the low-level list processing operations that are needed to update the data structures in steps A3 and A4, and to downdate them in A7 and A8.)

The move codes m_j of Algorithm A are integers between 0 and 5 that encode the state of the algorithm's progress as follows:

- $m_j = 0$ means we're trying $x_j = 1$ and haven't yet tried $x_j = 0$.
- $m_j = 1$ means we're trying $x_j = 0$ and haven't yet tried $x_j = 1$.
- $m_j = 2$ means we're trying $x_j = 1$ after $x_j = 0$ has failed.
- $m_j = 3$ means we're trying $x_j = 0$ after $x_j = 1$ has failed.
- $m_j = 4$ means we're trying $x_j = 1$ when $\bar{x}_j$ doesn't appear.
- $m_j = 5$ means we're trying $x_j = 0$ when x_j doesn't appear.

Codes 4 and 5 refer to so-called "pure literals": If no clause contains the literal $\bar{l}$, we can't go wrong by assuming that l is true.

For example, when Algorithm A is presented with the clauses (7), it cruises directly to a solution by setting $m_1m_2m_3m_4 = 1014$; the solution is $x_1x_2x_3x_4 = 0101$. But when the unsatisfiable clauses (6) are given, the successive code strings $m_1 \ldots m_d$ in step A2 are

$$1, 11, 110, 1131, 121, 1211, 1221, 21, 211, 2111, 2121, 221, 2221, \qquad (58)$$

before the algorithm gives up. (See Fig. 38.)

It's helpful to display the current string $m_1 \ldots m_d$ now and then, as a convenient indication of progress; this string increases lexicographically. Indeed, fascinating patterns appear as the 2s and 3s gradually move to the left. (Try it!)

When the algorithm terminates successfully in step A2, a satisfying assignment can be read off from the move table by setting $x_j \leftarrow 1 \oplus (m_j \mathbin{\&} 1)$ for $1 \le j \le d$. Algorithm A stops after finding a single solution; see exercise 122 if you want them all.

Lazy data structures. Instead of using the elaborate doubly linked machinery that underlies Algorithm A, we can actually get by with a much simpler scheme discovered by Cynthia A. Brown and Paul W. Purdom, Jr. [*IEEE Trans.* **PAMI-4** (1982), 309–316], who introduced the notion of *watched literals.* They observed that we don't really need to know all of the clauses that contain a given literal, because only one literal per clause is actually relevant at any particular time.

Here's the idea: When we work on clauses $F \mid L$, the variables that occur in L have known values, but the other variables do not. For example, in Algorithm A, variable x_j is implicitly known to be either true or false when $j \le d$, but its value is unknown when $j > d$. Such a situation is called a *partial assignment.* A partial assignment is *consistent* with a set of clauses if no clause consists entirely of false literals. Algorithms for SAT usually deal exclusively with consistent partial assignments; the goal is to convert them to consistent *total* assignments, by gradually eliminating the unknown values.

Thus every clause in a consistent partial assignment has at least one nonfalse literal; and we can assume that such a literal appears first, when the clause is represented in memory. Many nonfalse literals might be present, but only one of them is designated as the clause's "watchee." When a watched literal becomes false, we can find another nonfalse literal to swap into its place — unless the clause has been reduced to a unit, a clause of size 1.

With such a scheme we need only maintain a relatively short list for every literal l, namely a list W_l of all clauses that currently watch l. This list can be singly linked. Hence we need only one link per clause; and we have a total of only $2n + m$ links altogether, instead of the two links for each *cell* that are required by Algorithm A.

Furthermore — and this is the best part! — *no updates need to be made to the watch lists when backtracking.* The backtrack operations never falsify a nonfalse literal, because they only change values from known to unknown. Perhaps for this reason, data structures based on watched literals are called *lazy*, in contrast with the "eager" data structures of Algorithm A.

Let us therefore redesign Algorithm A and make it more laid-back. Our new data structure for each cell p has only one field, `L`(p); the other fields `F`(p), `B`(p), `C`(p) are no longer necessary, nor do we need $2n + 2$ special cells. As before we will represent clauses sequentially, with the literals of C_j beginning at `START`(j) for $1 \le j \le m$. The watched literal will be the one in `START`(j); and a new field, `LINK`(j), will be the number of another clause with the same watched literal (or 0, if C_j is the last such clause). Moreover, our new algorithm won't

lexicographically
lazy data structures–
Brown
Purdom
watched literals
partial assignment
consistent
unit
backtracking
eager

need SIZE(j). Instead, we can assume that the final literal of C_j is in location START($j-1$) $-$ 1, provided that we define START(0) appropriately.

The resulting procedure is almost unbelievably short and sweet. It's surely the simplest SAT solver that can claim to be efficient on problems of modest size:

move codes
pure literals
false literals preferred
unit clauses
Davis
Logemann

Algorithm B (*Satisfiability by watching*). Given nonempty clauses $C_1 \wedge \cdots \wedge C_m$ on $n > 0$ Boolean variables $x_1 \ldots x_n$, represented as above, this algorithm finds a solution if and only if the clauses are satisfiable. It records its current progress in an array $m_1 \ldots m_n$ of "moves," whose significance was explained above.

B1. [Initialize.] Set $d \leftarrow 1$.

B2. [Rejoice or choose.] If $d > n$, terminate successfully. Otherwise set $m_d \leftarrow [W_{2d} = 0 \text{ or } W_{2d+1} \neq 0]$ and $l \leftarrow 2d + m_d$.

B3. [Remove $\bar{l}$ if possible.] For all j such that $\bar{l}$ is watched in C_j, watch another literal of C_j. But go to B5 if that can't be done. (See exercise 124.)

B4. [Advance.] Set $W_{\bar{l}} \leftarrow 0$, $d \leftarrow d + 1$, and return to B2.

B5. [Try again.] If $m_d < 2$, set $m_d \leftarrow 3 - m_d$, $l \leftarrow 2d + (m_d \,\&\, 1)$, and go to B3.

B6. [Backtrack.] Terminate unsuccessfully if $d = 1$ (the clauses are unsatisfiable). Otherwise set $d \leftarrow d - 1$ and go back to B5. ▮

Readers are strongly encouraged to work exercise 124, which spells out the low-level operations that are needed in step B3. Those operations accomplish essentially everything that Algorithm B needs to do.

This algorithm doesn't use move codes 4 or 5, because lazy data structures don't have enough information to identify pure literals. Fortunately pure literals are comparatively unimportant in practice; problems that are helped by the pure literal shortcut can usually also be solved quickly without it.

Notice that steps A2 and B2 use different criteria for deciding whether to try $x_d = 1$ or $x_d = 0$ first at each branch of the search tree. Algorithm A chooses the alternative that satisfies the most clauses; Algorithm B chooses to make l true instead of $\bar{l}$ if the watch list for $\bar{l}$ is empty but the watch list for l is not. (All clauses in which $\bar{l}$ is watched will have to change, but those containing l are satisfied and in good shape.) In case of a tie, both algorithms set $m_d \leftarrow 1$, which corresponds to $x_d = 0$. The reason is that human-designed instances of SAT tend to have solutions made up of mostly false literals.

Forced moves from unit clauses. The simple logic of Algorithm B works well on many problems that aren't too large. But its insistence on setting x_1 first, then x_2, etc., makes it quite inefficient on many other problems, because it fails to take advantage of unit clauses. A unit clause (l) forces l to be true; therefore two-way branching is unnecessary whenever a unit clause is present. Furthermore, unit clauses aren't rare: Far from it. Experience shows that they're almost ubiquitous in practice, so that the actual search trees often involve only dozens of branch nodes instead of thousands or millions.

The importance of unit clauses was recognized already in the first computer implementation of a SAT solver, designed by Martin Davis, George Logemann,

and Donald Loveland [*CACM* **5** (1962), 394–397] and based on ideas that Davis had developed earlier with Hilary Putnam [*JACM* **7** (1960), 201–215]. They extended Algorithm A by introducing mechanisms that recognize when the size of a clause decreases to 1, or when the number of unsatisfied clauses containing a literal drops to 0. In such cases, they put variables onto a "ready list," and assigned those variables to fixed values before doing any further two-way branching. The resulting program was fairly complex; indeed, computer memory was so limited in those days, they implemented branching by writing all the data for the current node of the search tree onto magnetic tape, then backtracking when necessary by restoring the data from the most recently written tape records! The names of these four authors are now enshrined in the term "DPLL algorithm," which refers generally to SAT solving via partial assignment and backtracking.

Loveland
Davis
Putnam
pure literals
ready list
search tree
magnetic tape
backtracking
tape records
DPLL algorithm
Brown
Purdom
circular list
active ring
units
waerden

Brown and Purdom, in the paper cited earlier, showed that unit clauses can be detected more simply by using watched literals as in Algorithm B. We can supplement the data structures of that algorithm by introducing indices $h_1 \ldots h_n$ so that the variable whose value is being set at depth d is x_{h_d} instead of x_d. Furthermore we can arrange the not-yet-set variables whose watch lists aren't empty into a circular list called the "active ring"; the idea is to proceed through the active ring, checking to see whether any of its variables are currently in a unit clause. We resort to two-way branching only if we go all around the ring without finding any such units.

For example, let's consider the 32 unsatisfiable clauses of *waerden*(3, 3; 9) in (9). The active ring is initially (1 2 3 4 5 6 7), because 8, $\bar{8}$, 9, and $\bar{9}$ aren't being watched anywhere. There are no unit clauses yet. The algorithm below will decide to try $\bar{1}$ first; then it will change the clauses 123, 135, 147, and 159 to 213, 315, 417, and 519, respectively, so that nobody watches the false literal 1. The active ring becomes (2 3 4 5 6 7) and the next choice is $\bar{2}$; so 213, 234, 246, and 258 morph respectively into 312, 324, 426, 528. Now, with active ring (3 4 5 6 7), the unit clause '3' is detected (because 1 and 2 are false in '312'). This precipitates further changes, and the first steps of the computation can be summarized thus:

Active ring	$x_1x_2x_3x_4x_5x_6x_7x_8x_9$	Units	Choice	Changed clauses
(1 2 3 4 5 6 7)	- - - - - - - - -		$\bar{1}$	213, 315, 417, 519
(2 3 4 5 6 7)	0 - - - - - - - -		$\bar{2}$	312, 324, 426, 528
(3 4 5 6 7)	0 0 - - - - - - -	3	3	$\bar{4}3\bar{5}, \bar{5}3\bar{4}, \bar{6}3\bar{9}$
(4 5 6 7)	0 0 1 - - - - - -		$\bar{4}$	624, 714, 546, 648
(5 6 7)	0 0 1 0 - - - - -	6	6	$\bar{9}3\bar{6}, \bar{7}6\bar{8}$
(9 7 5)	0 0 1 0 - 1 - - -	$\bar{9}$	$\bar{9}$	
(7 5)	0 0 1 0 - 1 - - 0	7	7	$\bar{8}6\bar{7}, \bar{8}7\bar{9}$
(8 5)	0 0 1 0 - 1 1 - 0	$\bar{8}$	$\bar{8}$	
(5)	0 0 1 0 - 1 1 0 0	5, $\bar{5}$	Backtrack	
(6 9 7 8 5)	0 0 1 - - - - - -		4	$\bar{5}3\bar{4}, \bar{5}4\bar{6}, \bar{6}4\bar{8}$
(6 9 7 8 5)	0 0 1 1 - - - - -	$\bar{5}$	$\bar{5}$	456, 825, 915, 657, 759

(59)

When 6 is found, 7 is also a unit clause; but the algorithm doesn't see it yet, because variable x_6 is tested first. The active ring changes first to (7 5) after 6

empty list
cyclic DPLL
false literals preferred

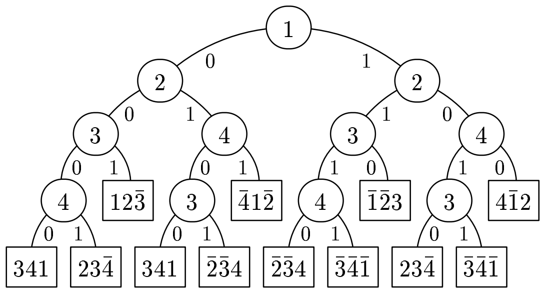


Fig. 39. The search tree that is implicitly traversed by Algorithm D, when that algorithm is applied to the eight unsatisfiable clauses R defined in (6). Branch nodes are labeled with the variable being tested; leaf nodes are labeled with a clause that is found to be contradicted. When the right child of a branch node is a leaf, the left branch was forced by a conditional unary clause.

is found, because 5 is cyclically after 6; we want to look at 7 before 5, instead of revisiting more or less the same clauses. After 6 has been chosen, 9 is inserted at the left, because the watch list for $\bar{9}$ becomes nonempty. After backtracking, variables 8, 7, 9, 6 are successively inserted at the left as they lose their forced values.

The following algorithm represents the active ring by giving a `NEXT` field to each variable, with $x_{\mathtt{NEXT}(k)}$ the successor of x_k. The ring is accessed via "head" and "tail" pointers h and t at the left and right, with $h = \mathtt{NEXT}(t)$. If the ring is empty, however, $t = 0$, and h is undefined.

Algorithm D (*Satisfiability by cyclic DPLL*). Given nonempty clauses $C_1 \wedge \cdots \wedge C_m$ on $n > 0$ Boolean variables $x_1 \ldots x_n$, represented with lazy data structures and an active ring as explained above, this algorithm finds a solution if and only if the clauses are satisfiable. It records its current progress in an array $h_1 \ldots h_n$ of indices and an array $m_0 \ldots m_n$ of "moves," whose significance is explained below.

D1. [Initialize.] Set $m_0 \leftarrow d \leftarrow h \leftarrow t \leftarrow 0$, and do the following for $k = n, n-1, \ldots, 1$: Set $x_k \leftarrow -1$ (denoting an unset value); if $W_{2k} \neq 0$ or $W_{2k+1} \neq 0$, set $\mathtt{NEXT}(k) \leftarrow h$, $h \leftarrow k$, and if $t = 0$ also set $t \leftarrow k$. Finally, if $t \neq 0$, complete the active ring by setting $\mathtt{NEXT}(t) \leftarrow h$.

D2. [Success?] Terminate if $t = 0$ (all clauses are satisfied). Otherwise set $k \leftarrow t$.

D3. [Look for unit clauses.] Set $h \leftarrow \mathtt{NEXT}(k)$ and use the subroutine in exercise 129 to compute $f \leftarrow [2h \text{ is a unit}] + 2[2h+1 \text{ is a unit}]$. If $f = 3$, go to D7. If $f = 1$ or 2, set $m_{d+1} \leftarrow f + 3$, $t \leftarrow k$, and go to D5. Otherwise, if $h \neq t$, set $k \leftarrow h$ and repeat this step.

D4. [Two-way branch.] Set $h \leftarrow \mathtt{NEXT}(t)$ and $m_{d+1} \leftarrow [W_{2h} = 0 \text{ or } W_{2h+1} \neq 0]$.

D5. [Move on.] Set $d \leftarrow d+1$, $h_d \leftarrow k \leftarrow h$. If $t = k$, set $t \leftarrow 0$; otherwise delete variable k from the ring by setting $\mathtt{NEXT}(t) \leftarrow h \leftarrow \mathtt{NEXT}(k)$.

D6. [Update watches.] Set $b \leftarrow (m_d + 1) \bmod 2$, $x_k \leftarrow b$, and clear the watch list for $\bar{x}_k$ (see exercise 130). Return to D2.

D7. [Backtrack.] Set $t \leftarrow k$. While $m_d \geq 2$, set $k \leftarrow h_d$, $x_k \leftarrow -1$; if $W_{2k} \neq 0$ or $W_{2k+1} \neq 0$, set `NEXT(k)` $\leftarrow h$, $h \leftarrow k$, `NEXT(t)` $\leftarrow h$; and set $d \leftarrow d - 1$.

D8. [Failure?] If $d > 0$, set $m_d \leftarrow 3 - m_d$, $k \leftarrow h_d$, and return to D6. Otherwise terminate the algorithm (because the clauses aren't satisfiable). ▮

move codes
nodes
search tree
langford(n)
Langford pairs
mems
pure literals

The move codes of this algorithm are slightly different from the earlier ones:

- $m_j = 0$ means we're trying $x_{h_j} = 1$ and haven't yet tried $x_{h_j} = 0$.
- $m_j = 1$ means we're trying $x_{h_j} = 0$ and haven't yet tried $x_{h_j} = 1$.
- $m_j = 2$ means we're trying $x_{h_j} = 1$ after $x_{h_j} = 0$ has failed.
- $m_j = 3$ means we're trying $x_{h_j} = 0$ after $x_{h_j} = 1$ has failed.
- $m_j = 4$ means we're trying $x_{h_j} = 1$ because it's forced by a unit clause.
- $m_j = 5$ means we're trying $x_{h_j} = 0$ because it's forced by a unit clause.

As before, the number of two-way branch nodes in the implicit search tree is the number of times that m_j is set to 0 or 1.

Comparison of the algorithms. OK, we've just seen three rudimentary SAT solvers. How well do they actually do? Detailed performance statistics will be given later in this section, after we've studied several more algorithms. But a brief quantitative study of Algorithms A, B, and D now will give us some concrete facts with which we can calibrate our expectations before moving on.

Consider, for example, *langford*(n), the problem of Langford pairs. This problem is typical of SAT instances where many unit clauses arise during the computation. For example, when Algorithm D is applied to *langford*(5), it reaches a stage where the move codes are

$$m_1 m_2 \ldots m_d \;=\; 125555555555555114545545, \qquad (60)$$

indicating only four two-way branches (the 1s and the 2) amongst a sea of forced moves. We therefore expect Algorithm D to outperform Algorithms A and B, which don't capitalize on unit clauses.

Sure enough, Algorithm D wins (slightly), even on a small example such as *langford*(5), which has 213 clauses, 480 cells, 28 variables. The detailed stats are

Algorithm A: $5379 + 108952$ mems, 10552 bytes, 705 nodes.
Algorithm B: $1206 + 30789$ mems, 4320 bytes, 771 nodes.
Algorithm D: $1417 + 28372$ mems, 4589 bytes, 11 nodes.

(Here "$5379 + 108952$ mems" means that 5379 memory accesses were made while initializing the data structures before the algorithm began; then the algorithm itself accessed octabytes of memory 108,952 times.) Notice that Algorithm B is more than thrice as fast as Algorithm A in this example, although it makes 771 two-way branches instead of 705. Algorithm A needs fewer nodes, because it recognizes pure literals; but Algorithm B does much less work per node. Algorithm D, on the other hand, works very hard at each node, yet comes out ahead because its decision-making choices reduce the search to only a few nodes.

benchmark test
variations in performance
waerden

These differences become more dramatic when we consider larger problems. For instance, *langford*(9) has 1722 clauses, 3702 cells, 104 variables, and we find

Algorithm A: 332.0 megamems, 77216 bytes, 1,405,230 nodes.
Algorithm B: 53.4 megamems, 31104 bytes, 1,654,352 nodes.
Algorithm D: 23.4 megamems, 32057 bytes, 6093 nodes.

And with *langford*(13)'s 5875 clauses, 12356 cells, 228 variables, the results are

Algorithm A: 2699.1 gigamems, 253.9 kilobytes, 8.7 giganodes.
Algorithm B: 305.2 gigamems, 101.9 kilobytes, 10.6 giganodes.
Algorithm D: 71.7 gigamems, 104.0 kilobytes, 14.0 meganodes.

Mathematicians will recall that, at the beginning of Chapter 7, we used elementary reasoning to prove the unsatisfiability of *langford*$(4k+1)$ for all k. Evidently SAT solvers have great difficulty discovering this fact, even when k is fairly small. We are using that problem here as a benchmark test, not because we recommend replacing mathematics by brute force! Its unsatisfiability actually enhances its utility as a benchmark, because algorithms for satisfiability are more easily compared with respect to unsatisfiable instances: Extreme variations in performance occur when clauses are satisfiable, because solutions can be found purely by luck. Still, we might as well see what happens when our three algorithms are set loose on the satisfiable problem *langford*(16), which turns out to be "no sweat." Its 11494 clauses, 23948 cells, and 352 variables lead to the statistics

Algorithm A: 11262.6 megamems, 489.2 kilobytes, 28.8 meganodes.
Algorithm B: 932.1 megamems, 196.2 kilobytes, 40.9 meganodes.
Algorithm D: 4.9 megamems, 199.4 kilobytes, 167 nodes.

Algorithm D is certainly our favorite so far, based on the *langford* data. But it is far from a panacea, because it loses badly to the lightweight Algorithm B on other problems. For example, the 2779 unsatisfiable clauses, 11662 cells, and 97 variables of *waerden*$(3,10;97)$ yield

Algorithm A: 150.9 gigamems, 212.8 kilobytes, 106.7114 meganodes.
Algorithm B: 6.2 gigamems, 71.2 kilobytes, 106.7116 meganodes.
Algorithm D: 1430.4 gigamems, 72.1 kilobytes, 102.7 meganodes.

And *waerden*$(3,10;96)$'s 2721 satisfiable clauses, 11418 cells, 96 variables give us

Algorithm A: 96.9 megamems, 208.3 kilobytes, 72.9 kilonodes.
Algorithm B: 12.4 megamems, 69.8 kilobytes, 207.7 kilonodes.
Algorithm D: 57962.8 megamems, 70.6 kilobytes, 4447.7 kilonodes.

In such cases unit clauses don't reduce the search tree size by very much, so we aren't justified in spending so much time per node.

***Speeding up by working harder.** Algorithms A, B, and D are OK on smallish problems, but they cannot really cope with the larger instances of SAT that have arisen in our examples. Significant enhancements are possible if we are willing to do more work and to develop more elaborate algorithms.

Mathematicians generally strive for nice, short, elegant proofs of theorems; and computer scientists generally aim for nice, short, elegant sequences of steps

with which a problem can quickly be solved. But some theorems have no short proofs, and some problems cannot be solved efficiently with short programs.

Let us therefore adopt a new attitude, at least temporarily, by fearlessly deciding to throw lots of code at SAT: Let's look at the bottlenecks that hinder Algorithm D on large problems, and let's try to devise new methods that will streamline the calculations even though the resulting program might be ten times larger. In this subsection we shall examine an advanced SAT solver, Algorithm L, which is able to outperform Algorithm D by many orders of magnitude on many important problems. This algorithm cannot be described in just a few lines; but it does consist of cooperating procedures that are individually nice, short, elegant, and understandable by themselves.

The first important ingredient of Algorithm L is an improved mechanism for unit propagation. Algorithm D needs only a few lines of code in step D3 to discover whether or not the value of an unknown variable has been forced by previous assignments; but that mechanism isn't particularly fast, because it is based on indirect inferences from a lazy data structure. We can do better by using "eager" data structures that are specifically designed to recognize forced values quickly, because high-speed propagation of the consequences of a newly asserted value turns out to be extremely important in practice.

A literal l is forced true when it appears in a clause C whose other literals have become false, namely when the set of currently assigned literals L has reduced C to the unit clause $C \,|\, L = (l)$. Such unit clauses arise from the reduction of binary clauses. Algorithm L therefore keeps track of the binary clauses $(u \vee v)$ that are relevant to the current subproblem $F \mid L$. This information is kept in a so-called "bimp table" $\texttt{BIMP}(l)$ for every literal l, which is a list of other literals l' whose truth is implied by the truth of l. Indeed, instead of simply including binary clauses within the whole list of given clauses, as Algorithms A, B, and D do, Algorithm L stores the relevant facts about $(u \vee v)$ directly, in a ready-to-use way, by listing u in $\texttt{BIMP}(\bar{v})$ and v in $\texttt{BIMP}(\bar{u})$. Each of the $2n$ tables $\texttt{BIMP}(l)$ is represented internally as a sequential list of length $\texttt{BSIZE}(l)$, with memory allocated dynamically via the buddy system (see exercise 134).

Binary clauses, in turn, are spawned by ternary clauses. For simplicity, Algorithm L assumes that all clauses have length 3 or less, because every instance of general SAT can readily be converted to 3SAT form (see exercise 28). And for speed, Algorithm L represents the ternary clauses by means of "timp tables," which are analogous to the bimp tables: Every literal l has a sequential list $\texttt{TIMP}(l)$ of length $\texttt{TSIZE}(l)$, consisting of pairs $p_1 = (u_1, v_1)$, $p_2 = (u_2, v_2)$, …, such that the truth of l implies that each $(u_i \vee v_i)$ becomes a relevant binary clause. If $(u \vee v \vee w)$ is a ternary clause, there will be three pairs $p = (v, w)$, $p' = (w, u)$, and $p'' = (u, v)$, appearing in the respective lists $\texttt{TIMP}(\bar{u})$, $\texttt{TIMP}(\bar{v})$, and $\texttt{TIMP}(\bar{w})$. Moreover, these three pairs are linked together cyclically, with

$$\texttt{LINK}(p) = p', \qquad \texttt{LINK}(p') = p'', \qquad \texttt{LINK}(p'') = p. \qquad (61)$$

Memory is allocated for the timp tables once and for all, as the clauses are input, because Algorithm L does not generate new ternaries during its computations.

unit propagation
lazy data structure
eager
unit clause
binary clauses
bimp table
sequential list
buddy system
ternary clauses
timp tables

Individual pairs p are, however, swapped around within these sequential tables, so that the currently active ternary clauses containing u always appear in the first TSIZE($\bar{u}$) positions that have been allocated to TIMP($\bar{u}$).

For example, let's consider again the ternary clauses (9) of *waerden*(3, 3; 9). Initially there are no binary clauses, so all BIMP tables are empty. Each of the ternary clauses appears in three of the TIMP tables. At level 0 of the search tree we might decide that $x_5 = 0$; then TIMP($\bar{5}$) tells us that we gain eight binary clauses, namely $\{13, 19, 28, 34, 37, 46, 67, 79\}$. These new binary clauses are represented by sixteen entries in BIMP tables; BIMP($\bar{3}$), for instance, will now be $\{1, 4, 7\}$. Furthermore, we'll want all of the TIMP pairs that involve either 5 or $\bar{5}$ to become inactive, because the ternary clauses that contain 5 are weaker than the new binary clauses, and the ternary clauses that contain $\bar{5}$ are now satisfied. (See exercise 136.)

As in (57) above, we shall assume that the variables of a given formula are numbered from 1 to n, and we represent the literals k and $\bar{k}$ internally by the numbers $2k$ and $2k+1$. Algorithm L introduces a new twist, however, by allowing variables to have many different *degrees of truth* [see M. Heule, M. Dufour, J. van Zwieten, and H. van Maaren, *LNCS* **3542** (2005), 345–359]: We say that x_k is true with degree D if VAL[k] $= D$, and false with degree D if VAL[k] $= D + 1$, where D is any even number.

The highest possible degree, typically $2^{32} - 2$ inside a computer, is called RT for "real truth." The next highest degree, typically $2^{32} - 4$, is called NT for "near truth"; and then comes PT $= 2^{32} - 6$, "proto truth." Lower degrees PT $- 2$, PT $- 4$, ..., 2 also turn out to be useful. A literal l is said to be *fixed in context* T if and only if VAL[$|l|$] $\geq T$; it is *fixed true* if we also have VAL[$|l|$] & $1 = l$ & 1, and it is *fixed false* if its complement $\bar{l}$ is fixed true.

Suppose, for example, that VAL[2] = RT + 1 and VAL[7] = PT; hence x_2 is "really false" while x_7 is "proto true." Then the literal '7', represented internally by $l = 14$, is fixed true in context PT, but l is not fixed in contexts NT or RT. The literal '$\bar{2}$', represented internally by $l = 5$, is fixed true in *every* context.

Algorithm L uses a sequential stack $R_0, R_1, \ldots,$ to record the names of literals that have received values. The current stack size, E, satisfies $0 \leq E \leq n$. With those data structures we can use a simple breadth-first search procedure to propagate the binary consequences of a literal l in context T at high speed:

Set $H \leftarrow E$; take account of l;
while $H < E$, set $l \leftarrow R_H$, $H \leftarrow H + 1$, and take account of l' for all l' in BIMP(l). (62)

Here "take account of l" means "if l is fixed true in context T, do nothing; if l is fixed false in context T, go to step CONFLICT; otherwise set VAL[$|l|$] $\leftarrow T + (l \& 1)$, $R_E \leftarrow l$, and $E \leftarrow E + 1$." The step called CONFLICT is changeable.

A literal's BIMP table might grow repeatedly as computation proceeds. But we can undo the consequences of bad decisions by simply resetting BSIZE(l) to the value that it had before those decisions were made. A special variable ISTAMP is increased whenever we begin a new round of decision-making, and each

waerden
degrees of truth
Heule
Dufour
van Zwieten
van Maaren
RT
real truth
NT
near truth
proto truth
PT
fixed literals
sequential
stack
breadth-first search
undo
ISTAMP

literal l has its private stamp IST(l). Whenever BSIZE(l) is about to increase, we check if IST(l) = ISTAMP. If not, we set

$$\texttt{IST}(l) \leftarrow \texttt{ISTAMP}, \quad \texttt{ISTACK}[I] \leftarrow \big(l, \texttt{BSIZE}(l)\big), \quad I \leftarrow I+1. \qquad (63)$$

Then the entries on ISTACK make it easy to downdate the BIMP tables when we backtrack. (See step L13 in the algorithm below.)

We're ready now to look at the detailed steps of Algorithm L, except that one more member of its arsenal of data structures needs to be introduced: There's an array VAR, which contains a permutation of $\{1,\ldots,n\}$, with VAR[k] $= x$ if and only if INX[x] $= k$. Furthermore VAR[k] is a "free variable" — not fixed in context RT — if and only if $0 \le k < N$. This setup makes it convenient to keep track of the variables that are currently free: A variable becomes fixed by swapping it to the end of the free list and decreasing N (see exercise 137); then we can free it later by simply increasing N, without swapping.

stamp
stamping
ISTAMP
ISTACK
IST
VAR
INX
free variable
DPLL with lookahead
really true
nearly true

Algorithm L (*Satisfiability by DPLL with lookahead*). Given nonempty clauses $C_1 \wedge \cdots \wedge C_m$ of size ≤ 3, on $n > 0$ Boolean variables $x_1 \ldots x_n$, this algorithm finds a solution if and only if the clauses are satisfiable. Its family of cooperating data structures is discussed in the text.

L1. [Initialize.] Record all binary clauses in the BIMP array and all ternary clauses in the TIMP array. Let U be the number of distinct variables in unit clauses; terminate unsuccessfully if two unit clauses contradict each other, otherwise record all distinct unit literals in FORCE[k] for $0 \le k < U$. Set VAR[k] $\leftarrow k+1$ and INX[$k+1$] $\leftarrow k$ for $0 \le k < n$; and $d \leftarrow F \leftarrow I \leftarrow$ ISTAMP $\leftarrow 0$. (Think d = depth, F = fixed variables, I = ISTACK size.)

L2. [New node.] Set BRANCH[d] $\leftarrow -1$. If $U = 0$, invoke Algorithm X below (which looks ahead for simplifications and also gathers data about how to make the next branch). Terminate happily if Algorithm X finds all clauses satisfied; go to L15 if Algorithm X discovers a conflict; go to L5 if $U > 0$.

L3. [Choose l.] Select a literal l that's desirable for branching (see exercise 168). If $l = 0$, set $d \leftarrow d+1$ and return to L2. Otherwise set DEC[d] $\leftarrow l$, BACKF[d] $\leftarrow F$, BACKI[d] $\leftarrow I$, and BRANCH[d] $\leftarrow 0$.

L4. [Try l.] Set $U \leftarrow 1$, FORCE[0] $\leftarrow l$.

L5. [Accept near truths.] Set $T \leftarrow$ NT, $G \leftarrow E \leftarrow F$, ISTAMP $\leftarrow$ ISTAMP $+ 1$, and CONFLICT $\leftarrow$ L11. Perform the binary propagation routine (62) for $l \leftarrow$ FORCE[0], ..., $l \leftarrow$ FORCE[$U-1$]; then set $U \leftarrow 0$.

L6. [Choose a nearly true L.] (At this point the stacked literals R_k are "really true" for $0 \le k < G$, and "nearly true" for $G \le k < E$. We want them all to be really true.) If $G = E$, go to L10. Otherwise set $L \leftarrow R_G$, $G \leftarrow G+1$.

L7. [Promote L to real truth.] Set $X \leftarrow |L|$ and VAL[X] $\leftarrow$ RT$+L\,\&\,1$. Remove variable X from the free list and from all TIMP pairs (see exercise 137). Do step L8 for all pairs (u,v) in TIMP(L), then return to L6.

L8. [Consider $u \vee v$.] (We have deduced that u or v must be true; five cases arise.) If either u or v is fixed true (in context T, which equals NT), do

compensation resolvents
langford
waerden
rand+
Algorithm L^0

nothing. If both u and v are fixed false, go to CONFLICT. If u is fixed false but v isn't fixed, perform (62) with $l \leftarrow v$. If v is fixed false but u isn't fixed, perform (62) with $l \leftarrow u$. If neither u nor v is fixed, do step L9.

L9. [Exploit $u \vee v$.] If $\bar{v} \in$ BIMP($\bar{u}$), perform (62) with $l \leftarrow u$ (because $\bar{u}$ implies both v and $\bar{v}$). Otherwise if $v \in$ BIMP($\bar{u}$), do nothing (because we already have the clause $u \vee v$). Otherwise if $\bar{u} \in$ BIMP($\bar{v}$), perform (62) with $l \leftarrow v$. Otherwise append v to BIMP($\bar{u}$) and u to BIMP($\bar{v}$). (Each change to BIMP means that (63) might be invoked. Exercise 139 explains how to improve this step by deducing further implications called "compensation resolvents.")

L10. [Accept real truths.] Set $F \leftarrow E$. If BRANCH[d] ≥ 0, set $d \leftarrow d+1$ and go to L2. Otherwise go to L3 if $d > 0$, to L2 if $d = 0$.

L11. [Unfix near truths.] While $E > G$, set $E \leftarrow E - 1$ and VAL[$|R_E|$] $\leftarrow 0$.

L12. [Unfix real truths.] While $E > F$, do the following: Set $E \leftarrow E - 1$ and $X \leftarrow |R_E|$; reactivate the TIMP pairs that involve X and restore X to the free list (see exercise 137); set VAL[X] $\leftarrow 0$.

L13. [Downdate BIMPs.] If BRANCH[d] ≥ 0, do the following while $I >$ BACKI[d]: Set $I \leftarrow I - 1$ and BSIZE(l) $\leftarrow s$, where ISTACK[I] $= (l, s)$.

L14. [Try again?] (We've discovered that DEC[d] doesn't work.) If BRANCH[d] $=$ 0, set $l \leftarrow$ DEC[d], DEC[d] $\leftarrow l \leftarrow \bar{l}$, BRANCH[$d$] $\leftarrow 1$, and go back to L4.

L15. [Backtrack.] Terminate unsuccessfully if $d = 0$. Otherwise set $d \leftarrow d - 1$, $E \leftarrow F$, $F \leftarrow$ BACKF[d], and return to L12. ▮

Exercise 143 extends this algorithm so that it will handle clauses of arbitrary size.

***Speeding up by looking ahead.** Algorithm L as it stands is incomplete, because step L2 relies on an as-yet-unspecified "Algorithm X" before choosing a literal for branching. If we use the simplest possible Algorithm X, by branching on whatever literal happens to be first in the current list of free variables, the streamlined methods for propagating forced moves in (62) and (63) will tend to make Algorithm L run roughly three times as fast as Algorithm D, and that isn't a negligible improvement. But with a sophisticated Algorithm X we can often gain another factor of 10 or more in speed, on significant problems.

For example, here are some typical empirical statistics:

Problem	Algorithm D	Algorithm L^0	Algorithm L^+
waerden(3, 10; 97)	1430 gigamems, 103 meganodes	391 gigamems, 31 meganodes	772 megamems, 4672 nodes
langford(13)	71.7 gigamems, 14.0 meganodes	21.5 gigamems, 10.9 meganodes	45.7 gigamems, 944 kilonodes
rand(3, 420, 100, 0)	184 megamems, 34 kilonodes	34 megamems, 7489 nodes	626 kilomems, 19 nodes

Here Algorithm L^0 stands for Algorithm L with the simplest Algorithm X, while Algorithm L^+ uses all of the lookahead heuristics that we are about to discuss. The first two problems involve rather large clauses, so they use the extended

Algorithm L of exercise 143. The third problem consists of 420 random ternary clauses on 100 variables. (Algorithm B, incidentally, needs 80.1 teramems, and a search tree of 4.50 teranodes, to show that those clauses are unsatisfiable.)

random ternary clauses
Heule
`march`
heuristic score

The moral of this story is that it's wise to do 100 times as much computation at every node of a large search tree, if we can thereby decrease the size of the tree by a factor of 1000.

How then can we distinguish a variable that's good for branching from a variable that isn't? We shall consider a three-step approach:

- Preselecting, to identify free variables that appear to be good candidates;
- Nesting, to allow candidate literals to share implied computations;
- Exploring, to examine the immediate consequences of hypothetical decisions.

While carrying out these steps, Algorithm X might discover a contradiction (in which case Algorithm L will take charge again at step L15); or the lookahead process might discover that several of the free literals are forced to be true (in which case it places them in the first U positions of the `FORCE` array). The explorations might even discover a way to satisfy all of the clauses (in which case Algorithm L will terminate and everybody will be happy). Thus, Algorithm X might do much more than simply choose a good variable on which to branch.

The following recommendations for Algorithm X are based on Marijn Heule's lookahead solver called `march`, one of the world's best, as it existed in 2013.

The first stage, preselection, is conceptually simplest, although it also involves some "handwaving" because it depends on necessarily shaky assumptions. Suppose there are N free variables. Experience has shown that we tend to get a good heuristic score $h(l)$ for each literal l, representing the relative amount by which asserting l will reduce the current problem, if these scores approximately satisfy the simultaneous nonlinear equations

$$h(l) = 0.1 + \alpha \sum_{\substack{u\in\mathtt{BIMP}(l)\\ u\text{ not fixed}}} \hat{h}(u) + \sum_{(u,v)\in\mathtt{TIMP}(l)} \hat{h}(u)\hat{h}(v). \qquad (64)$$

Here α is a magic constant, typically 3.5; and $\hat{h}(l)$ is a multiple of $h(l)$ chosen so that $\sum_l \hat{h}(l) = 2N$ is the total number of free literals. (In other words, the h scores on the right are "normalized" so that their average is 1.)

Any given set of scores $h(l)$ can be used to derive a refined set $h'(l)$ by letting

$$h'(l) = 0.1 + \alpha \sum_{\substack{u\in\mathtt{BIMP}(l)\\ u\text{ not fixed}}} \frac{h(u)}{h_{\rm ave}} + \sum_{(u,v)\in\mathtt{TIMP}(l)} \frac{h(u)}{h_{\rm ave}}\frac{h(v)}{h_{\rm ave}}, \quad h_{\rm ave} = \frac{1}{2N}\sum_l h(l). \qquad (65)$$

Near the root of the search tree, when $d \le 1$, we start with $h(l) = 1$ for all l and then refine it five times (say). At deeper levels we start with the $h(l)$ values from the parent node and refine them once. Exercise 145 contains an example.

We've computed $h(l)$ for all of the free literals l, but we won't have time to explore them all. The next step is to select free variables $\mathtt{CAND}[j]$ for $0 \le j < C$, where C isn't too large; we will insist that the number of candidates does not exceed

$$C_{\max} = \max(C_0, C_1/d), \qquad (66)$$

using cutoff parameters that are typically $C_0 = 30$, $C_1 = 600$. (See exercise 148.)

We start by dividing the free variables into "participants" and "newbies": A *participant* is a variable such that either x or $\bar{x}$ has played the role of u or v in step L8, at some node above us in the search tree; a *newbie* is a nonparticipant. When $d = 0$ every variable is a newbie, because we're at the root of the tree. But usually there is at least one participant, and we want to branch only on participants whenever possible, in order to maintain focus while backtracking.

If we've got too many potential candidates, even after restricting consideration to participants, we can winnow the list down by preferring the variables x that have the largest combined score $h(x)h(\bar{x})$. Step X3 below describes a fairly fast way to come up with the desired selection of $C \le C_{\max}$ candidates.

A simple lookahead algorithm can now proceed to compute a more accurate heuristic score $H(l)$, for each of the $2C$ literals $l = \texttt{CAND[}j\texttt{]}$ or $l = \neg\texttt{CAND[}j\texttt{]}$ that we've selected for further scrutiny. The idea is to simulate what would happen if l were used for branching, by mimicking steps L4–L9 (at least to a first approximation): Unit literals are propagated as in the exact algorithm, but whenever we get to the part of step L9 that changes the BIMP tables, we don't actually make such a change; we simply note that a branch on l would imply $u \vee v$, and we consider the value of that potential new clause to be $h(u)h(v)$. The heuristic score $H(l)$ is then defined to be the sum of all such clause weights:

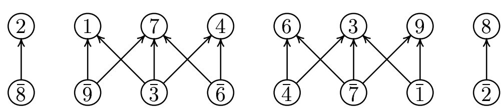
$$H(l) = \sum \bigl\{ h(u)h(v) \bigm| \text{asserting } l \text{ in L4 leads to asserting } u \vee v \text{ in L9} \bigr\}. \qquad (67)$$

For example, the problem *waerden*$(3,3;9)$ of (9) has nine candidate variables $\{1,2,\ldots,9\}$ at the root of the search tree, and exercise 145 finds their rough heuristic scores $h(l)$. The more discriminating scores $H(l)$ turn out to be

$$\begin{aligned}
H(1) &= h(2)h(3) + h(3)h(5) + h(4)h(7) + h(5)h(9) = 168.6;\\
H(2) &= h(1)h(3) + h(3)h(4) + h(4)h(6) + h(5)h(8) = 157.3;\\
H(3) &= h(1)h(2) + h(2)h(4) + h(4)h(5) + \cdots + h(6)h(9) = 233.4;\\
H(4) &= h(2)h(3) + h(3)h(5) + h(5)h(6) + \cdots + h(1)h(7) = 231.8;\\
H(5) &= h(3)h(4) + h(3)h(6) + h(6)h(7) + \cdots + h(1)h(9) = 284.0.
\end{aligned}$$

This problem is symmetrical, so we also have $H(6) = H(\bar{6}) = H(4) = H(\bar{4})$, etc. The best literal for branching, according to this estimate, is 5 or $\bar{5}$.

Suppose we set x_5 false and proceed to look ahead at the reduced problem, with $d = 1$. At this point there are eight candidates, $\{1,2,3,4,6,7,8,9\}$; and they're now related also by binary implications, because the original clause '357' has, for instance, been reduced to '37'. In fact, the BIMP tables now define the dependency digraph



(68)

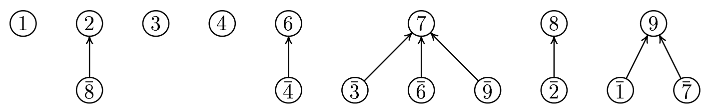
because $\bar{3} \longrightarrow 7$, etc.; and in general the $2C$ candidate literals will define a dependency digraph whose structure yields important clues about the current subproblem. We can, for example, use Tarjan's algorithm to find the strong

cutoff parameters
participants
newbies
focus
heuristic score
dependency digraph
binary implication graph, see dependency digrap
Tarjan
strong components

subforest
Tarjan
preorder
postorder
lookahead forest
tree-based lookahead, see lookahead forest
proto true
PT

components of that digraph, as mentioned after Theorem 7.1.1K. If some strong component includes both l and $\bar{l}$, the current subproblem is unsatisfiable. Otherwise two literals of the same component are constrained to have the same value; so we shall choose one literal from each of the $S \le 2C$ strong components, and use those choices as the actual candidates for lookahead.

Continuing our example, at this point we can use a nice trick to save redundant computation, by extracting a *subforest* of the dependency digraph:



(69)

The relation $\bar{8} \longrightarrow 2$ means that whatever happens after asserting the literal '2' will also happen after asserting '$\bar{8}$'; hence we need not repeat the steps for '2' while studying '$\bar{8}$'. And similarly, each of the other subordinate literals '$\bar{1}$', ..., '$\bar{9}$' inherits the assertions of its parent in this hierarchy. Tarjan's algorithm actually produces such a subforest with comparatively little extra work.

The nested structure of a forest also fits beautifully with "degrees of truth" in our data structure, if we visit the S candidate literals in *preorder* of the subforest, and if we successively assert each literal l at the truth degree that corresponds to twice its position in *postorder*. For instance, (69) becomes the following arrangement, which we shall call the "lookahead forest":

preorder	1	2	$\bar{8}$	3	4	6	$\bar{4}$	7	$\bar{3}$	$\bar{6}$	$\bar{9}$	8	$\bar{2}$	9	$\bar{1}$	$\bar{7}$
2·postorder	2	6	4	8	10	14	12	22	16	18	20	26	24	32	28	30

(70)

A simulation of steps L4–L9 with $l \leftarrow 1$ and $T \leftarrow 2$ makes x_1 true at degree 2 (we say that it's "2fixed" or "2true"); it also computes the score $H(1) \leftarrow h(\bar{2})h(\bar{3}) + h(\bar{4})h(\bar{7})$, but it spawns no other activity if Algorithm Y below isn't active. Simulation with $l \leftarrow 2$ and $T \leftarrow 6$ then 6fixes 2 and computes $H(2) \leftarrow h(\bar{1})h(\bar{3}) + h(\bar{4})h(\bar{6})$; during this process the value of x_1 isn't seen, because it is less than T. But things get more interesting when $l \leftarrow \bar{8}$ and $T \leftarrow 4$: Now we 4fix $\bar{8}$, and we're still able to see that x_2 is true because $6 > T$. So we save a little computation by inheriting $H(2)$ and setting $H(\bar{8}) \leftarrow H(2) + h(4)h(6) + h(6)h(7) + h(7)h(9)$.

The real action begins to break through a few steps later, when we set $l \leftarrow \bar{4}$ and $T \leftarrow 12$. Then (62) will 12fix not only $\bar{4}$ but also 3, since $\bar{4} \longrightarrow 3$; and the 12truth of 3 will soon take us to the simulated step L8 with $u = \bar{6}$ and $v = \bar{9}$. Aha: We 12fix $\bar{9}$, because 6 is 14true. Then we also 12fix the literals 7, 1, ..., and reach a contradiction. *This contradiction shows that branching on $\bar{4}$ will lead to a conflict*; hence the literal 4 must be true, if the current clauses are satisfiable.

Whenever the lookahead simulation of Algorithm X learns that some literal l must be true, as in this example, it places l on the `FORCE` list and makes l *proto true* (that is, true in context `PT`). A proto true literal will remain fixed true throughout this round of lookahead, because all relevant values of T will be less than `PT`. Later, Algorithm L will promote proto truth to near truth, and ultimately to real truth — unless a contradiction arises. (And in the case of *waerden*$(3, 3; 9)$, such a contradiction does in fact arise; see exercise 150.)

Why does the combination of preorder and postorder work so magically in (70)? It's because of a basic property of forests in general, which we noted for example in exercise 2.3.2–20: *If u and v are nodes of a forest, u is a proper ancestor of v if and only if u precedes v in preorder and u follows v in postorder.* Moreover, when we look ahead at candidate literals in this way, an important invariant relation is maintained on the R stack, namely that truth degrees never increase as we move from the bottom to the top:

$$\texttt{VAL}[|R_{j-1}|] \mid 1 \;\;\geq\;\; \texttt{VAL}[|R_j|], \qquad \text{for } 1 < j < E. \tag{71}$$

Real truths appear at the bottom, then near truths, then proto truth, etc. For example, the stack at one point in the problem above contains seven literals,

$$\begin{array}{rcccccccc} j & = & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ R_j & = & \bar{5} & 6 & \bar{4} & 3 & \bar{9} & 7 & 1 \\ \texttt{VAL}[|R_j|] & = & \texttt{RT}{+}1 & 14 & 13 & 12 & 13 & 12 & 12 \end{array} .$$

One consequence is that the current visibility of truth values matches the recursive structure by which false literals are purged from ternary clauses.

The second phase of Algorithm X, after preselection of candidates, is called "nesting," because it constructs a lookahead forest analogous to (70). More precisely, it constructs a sequence of literals $\texttt{LL}[j]$ and corresponding truth offsets $\texttt{LO}[j]$, for $0 \leq j < S$. It also sets up PARENT pointers to indicate the forest structure more directly; for example, with (69) we would have $\texttt{PARENT}(\bar{8}) = 2$ and $\texttt{PARENT}(2) = \Lambda$.

The third phase, "exploration," now does the real work. It uses the lookahead forest to evaluate heuristics $H(l)$ for the candidate literals — and also (if it's lucky) to discover literals whose values are forced.

The heart of the exploration phase is a breadth-first search based on steps L5, L6, and L8. This routine propagates truth values of degree T and also computes w, the weight of new binary clauses that would be spawned by branching on l:

$$\begin{array}{l} \text{Set } l_0 \leftarrow l,\ i \leftarrow w \leftarrow 0, \text{ and } G \leftarrow E \leftarrow F; \text{ perform (62)}; \\ \text{while } G < E, \text{ set } L \leftarrow R_G,\ G \leftarrow G+1, \text{ and} \\ \qquad \text{take account of } (u,v) \text{ for all } (u,v) \text{ in } \texttt{TIMP}(L); \\ \text{generate new binary clauses } (\bar{l}_0 \vee W_k) \text{ for } 0 \leq k < i. \end{array} \tag{72}$$

Here "take account of (u,v)" means "if either u or v is fixed true (in context T), do nothing; if both u and v are fixed false, go to CONFLICT; if u is fixed false but v isn't fixed, set $W_i \leftarrow v$, $i \leftarrow i+1$, and perform (62) with $l \leftarrow v$; if v is fixed false but u isn't fixed, set $W_i \leftarrow u$, $i \leftarrow i+1$, and perform (62) with $l \leftarrow u$; if neither u nor v is fixed, set $w \leftarrow w + h(u)h(v)$."

Explanation: A ternary clause of the form $\bar{L} \vee u \vee v$, where L is fixed true and u is fixed false as a consequence of l_0 being fixed true, is called a "windfall." Such clauses are good news, because they imply that the *binary* clause $\bar{l}_0 \vee v$ must be satisfied in the current subproblem. Windfalls are recorded on a stack called W, and appended to the BIMP database at the end of (72).

forests
ancestor
invariant relation
nesting
lookahead forest
exploration
breadth-first search
windfall

autarky principle
pure literal
touches
participants

The exploration phase also exploits an important fact called the *autarky principle*, which generalizes the notion of "pure literal" that we discussed above in connection with Algorithm A. An "autarky" for a SAT problem F is a set of strictly distinct literals $A = \{a_1, \ldots, a_t\}$ with the property that every clause of F either contains at least one literal of A or contains none of the literals of $\overline{A} = \{\bar{a}_1, \ldots, \bar{a}_t\}$. In other words, A satisfies every clause that A or $\overline{A}$ "touches."

An autarky is a self-sufficient system. Whenever A is an autarky, we can assume without loss of generality that all of its literals are actually true; for if F is satisfiable, the untouched clauses are satisfiable, and A tells us how to satisfy the touched ones. Step X9 of the following algorithm shows that we can detect certain autarkies easily while we're looking ahead.

Algorithm X (*Lookahead for Algorithm L*). This algorithm, which is invoked in step L2 of Algorithm L, uses the data structures of that algorithm together with additional arrays of its own to explore properties of the current subproblem. It discovers $U \geq 0$ literals whose values are forced, and puts them in the `FORCE` array. It terminates either by (i) satisfying all clauses; (ii) finding a contradiction; or (iii) computing heuristic scores $H(l)$ that will allow step L3 to choose a good literal for branching. In case (iii) it might also discover new binary clauses.

X1. [Satisfied?] If $F = n$, terminate happily (no variables are free).

X2. [Compile rough heuristics.] Set $N = n - F$ and use (65) to compute a rough score $h(l)$ for each free literal l.

X3. [Preselect candidates.] Let C be the current number of free variables that are "participants," and put them into the `CAND` array. If $C = 0$, set $C \leftarrow N$ and put *all* free variables into `CAND`; terminate happily, however, if all clauses are satisfied (see exercise 152). Give each variable x in `CAND` the rating $r(x) = h(x)h(\bar{x})$. Then while $C > 2C_{\max}$ (see (66)), delete all elements of `CAND` whose rating exceeds the mean rating; but terminate this loop if no elements are actually deleted. Finally, if $C > C_{\max}$, reduce C to $C_{\max}$ by retaining only top-ranked candidates. (See exercise 153.)

X4. [Nest the candidates.] Construct a lookahead forest, represented in `LL[`j`]` and `LO[`j`]` for $0 \leq j < S$ and by `PARENT` pointers (see exercise 155).

X5. [Prepare to explore.] Set $U' \leftarrow j' \leftarrow$ `BASE` $\leftarrow j \leftarrow 0$ and `CONFLICT` $\leftarrow$ X13.

X6. [Choose l for lookahead.] Set $l \leftarrow$ `LL[`j`]` and $T \leftarrow$ `BASE` $+$ `LO[`j`]`. Set $H(l) \leftarrow H($`PARENT(`l`)`$)$, where $H(\Lambda) = 0$. If l is not fixed in context T, go to X8. Otherwise, if l is fixed false but not proto false, do step X12 with $l \leftarrow \bar{l}$.

X7. [Move to next.] If $U > U'$, set $U' \leftarrow U$ and $j' \leftarrow j$. Then set $j \leftarrow j + 1$. If $j = S$, set $j \leftarrow 0$ and `BASE` $\leftarrow$ `BASE` $+ 2S$. Terminate normally if $j = j'$, or if $j = 0$ and `BASE` $+ 2S \geq$ `PT`. Otherwise return to X6.

X8. [Compute sharper heuristic.] Perform (72). Then if $w > 0$, set $H(l_0) \leftarrow H(l_0) + w$ and go to X10.

X9. [Exploit an autarky.] If $H(l_0) = 0$, do step X12 with $l \leftarrow l_0$. Otherwise generate the new binary clause $l_0 \vee \neg$`PARENT(`l_0`)`. (Exercise 166 explains why.)

necessary assignments
forced literals
double truth
DT
Dtrue
Dfalse

X10. [Optionally look deeper.] Perform Algorithm Y below.

X11. [Exploit necessary assignments.] Do step X12 for all literals $l \in$ `BIMP`$(\bar{l}_0)$ that are fixed true but not proto true. Then go to X7. (See exercise 167.)

X12. [Force l.] Set `FORCE`$[U] \leftarrow l$, $U \leftarrow U + 1$, $T' \leftarrow T$, and perform (72) with $T \leftarrow$ `PT`. Then set $T \leftarrow T'$. (This step is a subroutine, used by other steps.)

X13. [Recover from conflict.] If $T <$ `PT`, do step X12 with $l \leftarrow \bar{l}_0$ and go to X7. Otherwise terminate with a contradiction. ▌

Notice that, in steps X5–X7, this algorithm proceeds cyclically through the forest, continuing to look ahead until completing a pass in which no new forced literals are found. The `BASE` address of truth values continues to grow, if necessary, but it isn't allowed to become too close to `PT`.

***Looking even further ahead.** If it's a good idea to look one step ahead, maybe it's a better idea to look *two* steps ahead. Of course that's a somewhat scary proposition, because our data structures are already pretty stretched; and besides, double lookahead might take way too much time. Nevertheless, there's a way to pull it off, and to make Algorithm L run even faster on many problems.

Algorithm X looks at the immediate consequences of assuming that some literal l_0 is true. Algorithm Y, which is launched in step X10, goes further out on that limb, and investigates what would happen if *another* literal, $\hat{l}_0$, were also true. The goal is to detect branches that die off early, allowing us to discover new implications of l_0 or even to conclude that l_0 must be false.

For this purpose Algorithm Y stakes out an area of truth space between the current context T and a degree of truth called "double truth" or `DT`, which is defined in step Y2. The size of this area is determined by a parameter Y, which is typically less than 10. The same lookahead forest is used to give relative truth degrees below `DT`. Double truth is less trustworthy than proto truth, `PT`; but literals that are fixed at level `DT` are known to be conditionally true ("Dtrue") or conditionally false ("Dfalse") under the hypothesis that l_0 is true.

Going back to our example of *waerden*$(3, 3; 9)$, the scenario described above was based on the assumption that double lookahead was not done. Actually, however, further activity by Algorithm Y will usually take place after $H(1)$ has been set to $h(\bar{2})h(\bar{3}) + h(\bar{4})h(\bar{7})$. The value of `DT` will be set to 130, assuming that $Y = 8$, because $S = 8$. Literal 1 will become Dtrue. Looking then at 2 will 6fix 2; and that will 6fix $\bar{3}$ because of the clause $\bar{1}\bar{2}\bar{3}$. Then $\bar{3}$ will 6fix 4 and 7, contradicting $\bar{1}\bar{4}\bar{7}$ and causing 2 to become Dfalse. Other literals also will soon become Dtrue or Dfalse, leading to a contradiction; and that contradiction will allow Algorithm Y to make literal 1 proto false before Algorithm X has even begun to look ahead at literal 2.

The main loop of double lookahead is analogous to (72), but it's simpler, because we're further removed from reality:

$$\begin{aligned}&\text{Set } \hat{l}_0 \leftarrow l \text{ and } G \leftarrow E \leftarrow F\text{; perform (62);}\\ &\text{while } G < E\text{, set } L \leftarrow R_G,\ G \leftarrow G + 1\text{, and}\\ &\quad\text{take account of } (u, v) \text{ for all } (u, v) \text{ in } \texttt{TIMP}(L).\end{aligned} \tag{73}$$

Now "take account of (u,v)" means "if either u or v is fixed true (in context T), or if neither u nor v is fixed, do nothing; if both u and v are fixed false, go to `CONFLICT`; if u is fixed false but v isn't fixed, perform (62) with $l \leftarrow v$; if v is fixed false but u isn't fixed, perform (62) with $l \leftarrow u$."

Since double-looking is costly, we want to try it only when there's a fairly good chance that it will be helpful, namely when $H(l_0)$ is large. But how large is large enough? The proper threshold depends on the problem being solved: Some sets of clauses are handled more quickly by double-looking, while others are immune to such insights. Marijn Heule and Hans van Maaren [*LNCS* **4501** (2007), 258–271] have developed an elegant feedback mechanism that automatically tunes itself to the characteristics of the problem at hand: Let τ be a "trigger," initially 0. Step Y1 allows double-look only if $H(l_0) > \tau$; otherwise τ is decreased to $\beta\tau$, where β is a damping factor (typically 0.999), so that double-looking will become more attractive. On the other hand if double-look doesn't find a contradiction that makes l_0 proto false, the trigger is raised to $H(l_0)$ in step Y6.

Heule
van Maaren
feedback mechanism
adaptive control
trigger
damping factor
DFAIL
pi
random ternary clauses

Algorithm Y (*Double lookahead for Algorithm X*). This algorithm, invoked in step X10, uses the same data structures (and a few more) to look ahead more deeply. Parameters β and Y are explained above. Initially `DFAIL`$(l) = 0$ for all l.

Y1. [Filter.] Terminate if `DFAIL`(l_0) = `ISTAMP`, or if $T + 2S(Y+1) >$ `PT`. Otherwise, if $H(l_0) \le \tau$, set $\tau \leftarrow \beta\tau$ and terminate.

Y2. [Initialize.] Set `BASE` $\leftarrow T-2$, `LBASE` $\leftarrow$ `BASE` $+ 2S\cdot Y$, `DT` $\leftarrow$ `LBASE` + `LO`$[\jmath]$, $i \leftarrow \hat\jmath' \leftarrow \hat\jmath \leftarrow 0$, $E \leftarrow F$, and `CONFLICT` $\leftarrow$ Y8. Perform (62) with $l \leftarrow l_0$ and $T \leftarrow$ `DT`.

Y3. [Choose l for double look.] Set $l \leftarrow$ `LL`$[\jmath]$ and $T \leftarrow$ `BASE` + `LO`$[\jmath]$. If l is not fixed in context T, go to Y5. Otherwise, if l is fixed false but not Dfalse, do step Y7 with $l \leftarrow \bar l$.

Y4. [Move to next.] Set $\hat\jmath \leftarrow \hat\jmath + 1$. If $\hat\jmath = S$, set $\hat\jmath \leftarrow 0$ and `BASE` $\leftarrow$ `BASE` $+ 2S$. Go to Y6 if $\hat\jmath' = \hat\jmath$, or if $\hat\jmath = 0$ and `BASE` = `LBASE`. Otherwise return to Y3.

Y5. [Look ahead.] Perform (73), and return to Y4 (if no conflict arises).

Y6. [Finish.] Generate new binary clauses $(\bar l_0 \vee W_k)$ for $0 \le k < i$. Then set `BASE` $\leftarrow$ `LBASE`, $T \leftarrow$ `DT`, $\tau \leftarrow H(l_0)$, `DFAIL`$(l_0) \leftarrow$ `ISTAMP`, `CONFLICT` $\leftarrow$ X13, and terminate.

Y7. [Make $\hat l_0$ false.] Set $\hat\jmath' \leftarrow \hat\jmath$, $T' \leftarrow T$, and perform (73) with $l \leftarrow \hat l_0$ and $T \leftarrow$ `DT`. Then set $T \leftarrow T'$, $W_i \leftarrow \hat l_0$, $i \leftarrow i+1$. (This step is a subroutine.)

Y8. [Recover from conflict.] If $T <$ `DT`, do step Y7 with $l \leftarrow \neg$`LL`$[\hat\jmath]$ and go to Y4. Otherwise set `CONFLICT` $\leftarrow$ X13 and exit to X13. ▮

Some quantitative statistics will help to ground these algorithms in reality: When Algorithm L was let loose on $rand(3, 2062, 500, 314)$, a problem with 500 variables and 2062 random ternary clauses, it proved unsatisfiability after making 684,433,234,661 memory accesses and constructing a search tree of 9,530,489 nodes. Exercise 173 explains what would have happened if various parts of the algorithm had been disabled. None of the other SAT solvers we shall discuss are able to handle such random problems in a reasonable amount of time.

random satisfiability–
3SAT–
probability of satisfiability–

Random satisfiability. There seems to be no easy way to analyze the satisfiability problem under random conditions. In fact, the basic question "How many random clauses of 3SAT on n variables do we need to consider, on the average, before they can't all be satisfied?" is a famous unsolved research problem.

From a practical standpoint this question isn't as relevant as the analogous questions were when we studied algorithms for sorting or searching, because real-world instances of 3SAT tend to have highly *non*random clauses. Deviations from randomness in combinatorial algorithms often have a dramatic effect on running time, while methods of sorting and searching generally stay reasonably close to their expected behavior. Thus a focus on randomness can be misleading. On the other hand, random SAT clauses do serve as a nice, clean model, so they give us insights into what goes on in Boolean territory. Furthermore the mathematical issues are of great interest in their own right. And fortunately, much of the basic theory is in fact elementary and easy to understand. So let's take a look at it.

Exercise 180 shows that random satisfiability can be analyzed *exactly*, when there are at most five variables. We might as well start there, because the "tiny" 5-variable case is still large enough to shed some light on the bigger picture. When there are n variables and k literals per clause, the number N of possible clauses that involve k different variables is clearly $2^k\binom{n}{k}$: There are $\binom{n}{k}$ ways to choose the variables, and 2^k ways to either complement or not. So we have, for example, $N = 2^3\binom{5}{3} = 80$ possible clauses in a 3SAT problem on 5 variables.

Let q_m be the probability that m of those clauses, distinct but otherwise selected at random, are satisfiable. Thus $q_m = Q_m/\binom{N}{m}$, where Q_m is the number of ways to choose m of the N clauses so that at least one Boolean vector $x = x_1 \ldots x_n$ satisfies them all. Figure 40 illustrates these probabilities when $k = 3$ and $n = 5$. Suppose we're being given distinct random clauses one by one. According to Fig. 40, the chances are better than 77% that we'll still be able to satisfy them after 20 different clauses have been received, because $q_{20} \approx 0.776$. But by the time we've accumulated 30 of the 80 clauses, the chance of satisfiability has dropped to $q_{30} \approx 0.179$; and after ten more we reach $q_{40} \approx 0.016$.

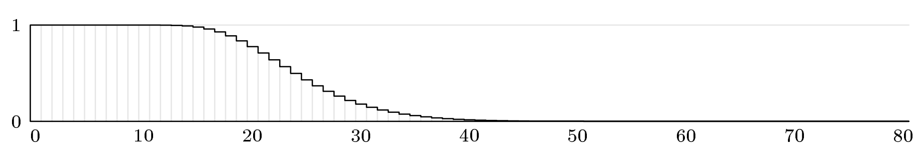


Fig. 40. The probability q_m that m distinct clauses of 3SAT on 5 variables are simultaneously satisfiable, for $0 \le m \le 80$.

The illustration makes it appear as if $q_m = 1$ for $m < 15$, say, and as if $q_m = 0$ for $m > 55$. But q_8 is actually *less* than 1, because of (6); exercise 179 gives the exact value. And q_{70} is *greater* than 0, because $Q_{70} = 32$; indeed, every Boolean vector x satisfies exactly $(2^k-1)\binom{n}{k} = (1-2^{-k})N$ of the N possible k-clauses, so it's no surprise that 70 noncontradictory 3-clauses on 5 variables can be found. Of course those clauses will hardly ever be the first 70 received, in a random situation. The actual value of q_{70} is $32/1646492110120 \approx 2 \times 10^{-11}$.

solutions, number of+
standard deviation
uniquely satisfiable
stopping time
summation by parts

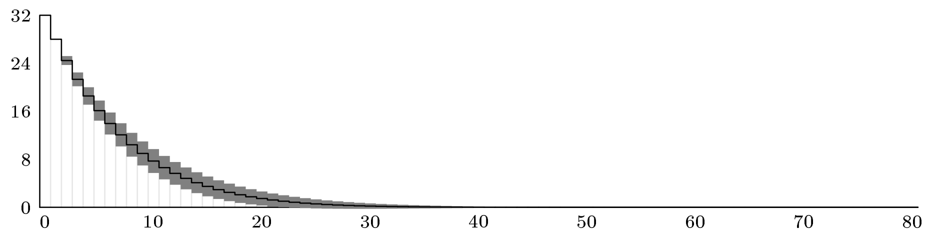


Fig. 41. The total number T_m of different Boolean vectors $x = x_1 \ldots x_5$ that simultaneously satisfy m distinct clauses of 3SAT on 5 variables, for $0 \le m \le 80$.

Figure 41 portrays the same process from another standpoint: It shows *in how many ways* a random set of m clauses can be satisfied. This value, T_m, is a random variable whose mean is indicated in black, surrounded by a gray region that shows the mean plus-or-minus the standard deviation. For example, T_0 is always 32, and T_1 is always 28; but T_2 is either 24, 25, or 26, and it takes these values with the respective probabilities $(2200, 480, 480)/3160$. Thus the mean for $m = 2$ is ≈ 24.5, and the standard deviation is ≈ 0.743.

When $m = 20$, we know from Fig. 40 that T_{20} is nonzero more than 77% of the time; yet Fig. 41 shows that $T_{20} \approx 1.47 \pm 1.17$. (Here the notation $\mu \pm \sigma$ stands for the mean value μ with standard deviation σ.) It turns out, in fact, that 20 random clauses are *uniquely satisfiable*, with $T_{20} = 1$, more than 33% of the time; and the probability that $T_{20} > 4$ is only 0.013. With 30 clauses, satisfiability gets dicier and dicier: $T_{30} \approx 0.20 \pm 0.45$; indeed, T_{30} is less than 2, more than 98% of the time — although it can be as high as 11 if the clause-provider is being nice to us. By the time 40 clauses are reached, the odds that T_{40} exceeds 1 are less than 1 in 4700. Figure 42 shows the probability that $T_m = 1$ as m varies.

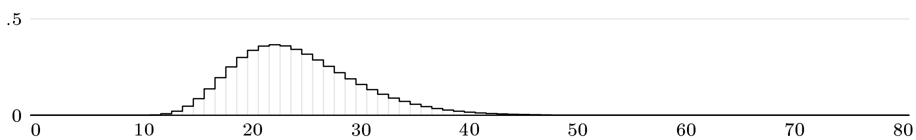


Fig. 42. $\Pr(T_m = 1)$, the probability that m distinct clauses of 3SAT on 5 variables are uniquely satisfiable, for $0 \le m \le 80$.

Let P be the number of clauses that have been received when we're first unable to satisfy them all. Thus we have $P = m$ with probability p_m, where $p_m = q_{m-1} - q_m$ is the probability that $m-1$ random clauses are satisfiable but m are not. These probabilities are illustrated in Fig. 43. Is it surprising that Figs. 42 and 43 look roughly the same? (See exercise 183.)

The expected "stopping time," $\mathrm{E}\,P$, is by definition equal to $\sum_m mp_m$; and it's not difficult to see, for example by using the technique of summation by parts (exercise 1.2.7–10), that we can compute it by summing the probabilities in Fig. 40:

$$\mathrm{E}\,P \;=\; \sum_m q_m. \tag{74}$$

variance
kSAT
4SAT
2SAT
1SAT
sampling with and without replacement
repeated clauses
birthday paradox

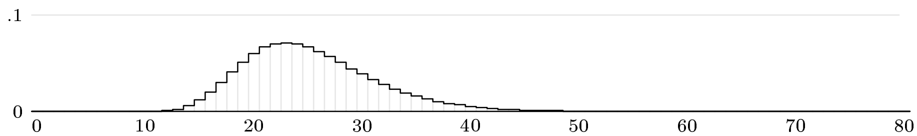


Fig. 43. The stopping time probabilities, p_m, that m distinct clauses of 3SAT on 5 variables have just become unsatisfiable, for $0 \le m \le 80$.

The variance of P, namely $\mathrm{E}(P - \mathrm{E}\,P)^2 = (\mathrm{E}\,P^2) - (\mathrm{E}\,P)^2$, also has a simple expression in terms of the q's, because

$$\mathrm{E}\,P^2 \;=\; \sum_m (2m+1)q_m. \tag{75}$$

In Figs. 40 and 43 we have $\mathrm{E}\,P \approx 25.22$, with variance ≈ 35.73.

So far we've been focusing our attention on 3SAT problems, but the same ideas apply also to kSAT for other clause sizes k. Figure 44 shows exact results for the probabilities when $n = 5$ and $1 \le k \le 4$. Larger values of k give clauses that are easier to satisfy, so they increase the stopping time. With five variables the typical stopping times for random 1SAT, 2SAT, 3SAT, and 4SAT turn out to be respectively 4.06 ± 1.19, 11.60 ± 3.04, 25.22 ± 5.98, and 43.39 ± 7.62. In general if $P_{k,n}$ is the stopping time for kSAT on n variables, we let

$$S_{k,n} \;=\; \mathrm{E}\,P_{k,n} \tag{76}$$

be its expected value.

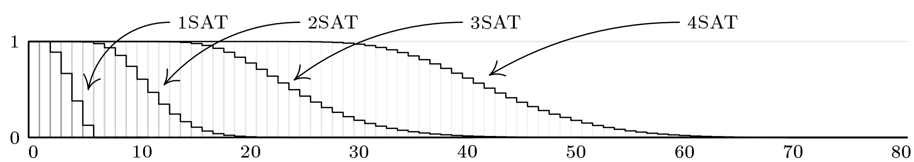


Fig. 44. Extension of Fig. 40 to clauses of other sizes.

Our discussions so far have been limited in another way too: We've been assuming that m *distinct* clauses are being presented to a SAT solver for solution. In practice, however, it's much easier to generate clauses by allowing repetitions, so that every clause is chosen without any dependence on the past history. In other words, there's a more natural way to approach random satisfiability, by assuming that N^m possible *ordered sequences* of clauses are equally likely after m steps, not that we have $\binom{N}{m}$ equally likely *sets* of clauses.

Let $\hat{q}_m$ be the probability that m random clauses $C_1 \wedge \cdots \wedge C_m$ are satisfiable, where each C_j is randomly chosen from among the $N = 2^k\binom{n}{k}$ possibilities in a kSAT problem on n variables. Figure 45 illustrates these probabilities in the case $k = 3$, $n = 5$; notice that we always have $\hat{q}_m \ge q_m$. If N is large while m is small, it's clear that $\hat{q}_m$ will be very close to q_m, because repeated clauses are unlikely in such a case. Still, we must keep in mind that q_N is always zero, while $\hat{q}_m$ is *never* zero. Furthermore, the "birthday paradox" discussed in Section 6.4 warns

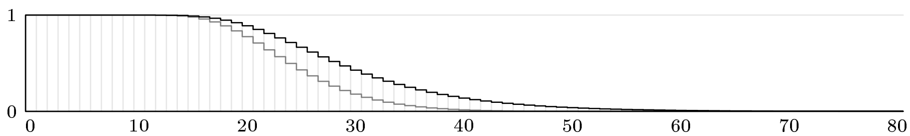


Fig. 45. Random 3SAT on 5 variables when the clauses are sampled with replacement. The probabilities $\hat{q}_m$ are shown with a black line; the smaller probabilities q_m of Fig. 40 are shown in gray.

us that repetitions aren't as rare as we might expect. The deviations of $\hat{q}_m$ from q_m are particularly noticeable in small cases such as the scenario of Fig. 45.

In any event, there's a direct way to compute $\hat{q}_m$ from the probabilities q_t and the value of N (see exercise 184):

$$\hat{q}_m \;=\; \sum_{t=0}^{N} \left\{ {m \atop t} \right\} t!\, q_t \binom{N}{t} \Big/ N^m. \tag{77}$$

And there are surprisingly simple formulas analogous to (74) and (75) for the stopping time $\widehat{P}$, where $\hat{p}_m = \hat{q}_{m-1} - \hat{q}_m$, as shown in exercise 186:

$$\mathrm{E}\,\widehat{P} = \sum_{m=0}^{N-1} \frac{N}{N-m}\, q_m; \tag{78}$$

$$\mathrm{E}\,\widehat{P}^2 = \sum_{m=0}^{N-1} \frac{N}{N-m}\, q_m \left(1 + 2\Big(\frac{N}{N-1} + \cdots + \frac{N}{N-m}\Big)\right). \tag{79}$$

These formulas prove that the expected behavior of $\widehat{P}$ is very much like that of P, if q_m is small whenever m/N isn't small. In the case $k=3$ and $n=5$, the typical stopping times $\widehat{P} = 30.58 \pm 9.56$ are significantly larger than those of P; but we are mostly interested in cases where n is large and where $\hat{q}_m$ is essentially indistinguishable from q_m. In order to indicate plainly that the probability $\hat{q}_m$ depends on k and n as well as on m, we shall denote it henceforth by $S_k(m,n)$:

$$S_k(m,n) = \Pr(m \text{ random clauses of } k\text{SAT are satisfiable}), \tag{80}$$

where the m clauses are "sampled with replacement" (they needn't be distinct). Suitable pseudorandom clauses $\mathit{rand}(k,m,n,\mathit{seed})$ can easily be generated.

Exact formulas appear to be out of reach when $n > 5$, but we can make empirical tests. For example, extensive experiments on random 3SAT problems by B. Selman, D. G. Mitchell, and H. J. Levesque [*Artificial Intelligence* **81** (1996), 17–29] showed a dramatic drop in the chances of satisfiability when the number of clauses exceeds about $4.27n$. This "phase transition" becomes much sharper as n grows (see Fig. 46).

Similar behavior occurs for random kSAT, and this phenomenon has spawned an enormous amount of research aimed at evaluating the so-called *satisfiability thresholds*

$$\alpha_k \;=\; \lim_{n\to\infty} S_{k,n}/n. \tag{81}$$

kSAT
sampled with replacement
rand
Selman
Mitchell
Levesque
phase transition
density of clauses: the number of clauses per var
crossover point, see threshold
satisfiability thresholds

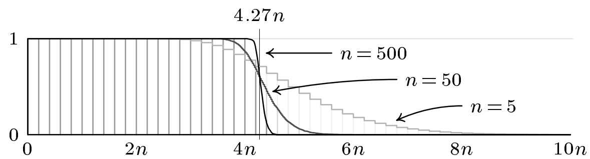


Fig. 46. Empirical probability data shows that random 3SAT problems rapidly become unsatisfiable when there are more than $\alpha_3 n$ clauses, if n is large enough.

Indeed, we can obtain quite difficult kSAT problems by generating approximately $\alpha_k n$ random k-clauses, using empirically observed estimates of α_k. If n is large, the running time for random 3SAT with $4.3n$ clauses is typically orders of magnitude larger than it is when the number of clauses is $4n$ or $4.6n$. (And still tougher problems arise in rare instances when we have, say, $3.9n$ clauses that happen to be *un*satisfiable.)

Strictly speaking, however, nobody has been able to prove that the so-called constants α_k actually exist, for all k! The empirical evidence is overwhelming; but rigorous proofs for $k = 3$ have so far only established the bounds

$$\liminf_{n\to\infty} S_{3,n}/n \ge 3.52; \qquad \limsup_{n\to\infty} S_{3,n}/n \le 4.49. \tag{82}$$

[See A. C. Kaporis, L. M. Kirousis, and E. G. Lalas, *Random Structures & Algorithms* **28** (2006), 444–480; J. Díaz, L. Kirousis, D. Mitsche, and X. Pérez-Giménez, *Theoretical Comp. Sci.* **410** (2009), 2920–2934.] A "sharp threshold" result has been established by E. Friedgut [*J. Amer. Math. Soc.* **12** (1999), 1017–1045, 1053–1054], who proved the existence for $k \ge 2$ of functions $\alpha_k(n)$ with

$$\lim_{n\to\infty} S_k\bigl(\lfloor(\alpha_k(n)-\epsilon)n\rfloor, n\bigr) = 1, \qquad \lim_{n\to\infty} S_k\bigl(\lfloor(\alpha_k(n)+\epsilon)n\rfloor, n\bigr) = 0, \tag{83}$$

when ϵ is any positive number. But those functions might not approach a limit. They might, for example, fluctuate periodically, like the "wobble function" that we encountered in Eq. 5.2.2–(47).

The current best guess for α_3, based on heuristics of the "survey propagation" technique to be discussed below, is that $\alpha_3 = 4.26675\pm0.00015$ [S. Mertens, M. Mézard, and R. Zecchina, *Random Structures & Algorithms* **28** (2006), 340–373]. Similarly, it appears reasonable to believe that $\alpha_4 \approx 9.931$, $\alpha_5 \approx 21.12$, $\alpha_6 \approx 43.37$, $\alpha_7 \approx 87.79$. The α's grow as $\Theta(2^k)$ (see exercise 195); and they *are* known to be constant when k is sufficiently large [see J. Ding, A. Sly, and N. Sun, *STOC* **47** (2015), to appear].

Analysis of random 2SAT. Although nobody knows how to prove that random 3SAT problems almost always become unsatisfiable when the number of clauses reaches $\approx 4.27n$, the corresponding question for 2SAT does have a nice answer: *The satisfiability threshold* α_2 *equals* 1. For example, when the author first tried 1000 random 2SAT problems with a million variables, 999 of them turned out to be satisfiable when there were 960,000 clauses, while all were unsatisfiable when the number of clauses rose to 1,040,000. Figure 47 shows how this transition becomes sharper as n increases.

Kaporis
Kirousis
Lalas
Díaz
Mitsche
Pérez-Giménez
sharp threshold
Friedgut
wobble function
survey propagation
Mertens
Mézard
Zecchina
4SAT
5SAT
6SAT
7SAT
kSAT
Ding
Sly
Sun
2SAT–

Chvátal
Reed
Goerdt
Fernandez de la Vega
implication digraph
Karp
giant strong component
strong components
Chvátal
Reed
s-chain
strictly distinct literals

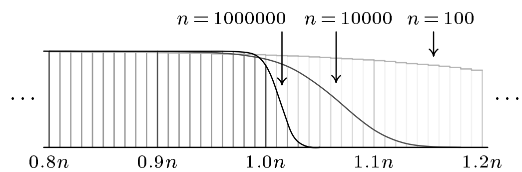


Fig. 47. Empirical satisfaction probabilities for 2SAT with approximately n random clauses. (When $n = 100$, the probability doesn't become negligible until more than roughly 180 clauses have been generated.)

The fact that $S_{2,n} \approx n$ was discovered in 1991 by V. Chvátal and B. Reed [*FOCS* **33** (1992), 620–627], and the same result was obtained independently at about the same time by A. Goerdt and by W. Fernandez de la Vega [see *J. Comp. Syst. Sci.* **53** (1996), 469–486; *Theor. Comp. Sci.* **265** (2001), 131–146].

The study of this phenomenon is instructive, because it relies on properties of the digraph that characterizes all instances of 2SAT. Furthermore, the proof below provides an excellent illustration of the "first and second moment principles," equations MPR–(21) and MPR–(22). Armed with those principles, we're ready to derive the 2SAT threshold:

Theorem C. *Let c be a fixed constant. Then*

$$\lim_{n\to\infty} S_2\bigl(\lfloor cn\rfloor, n\bigr) \;=\; \begin{cases} 1, & \text{if } c<1;\\ 0, & \text{if } c>1.\end{cases} \tag{84}$$

Proof. Every 2SAT problem corresponds to an *implication digraph* on the literals, with arcs $\bar{l} \longrightarrow l'$ and $\bar{l}' \longrightarrow l$ for each clause $l \vee l'$. We know from Theorem 7.1.1K that a set of 2SAT clauses is satisfiable if and only if no strong component of its implication digraph contains both x and $\bar{x}$ for some variable x. That digraph has $2m = 2\lfloor cn\rfloor$ arcs and $2n$ vertices. If it were a random digraph, well-known theorems of Karp (which we shall study in Section 7.4.1) would imply that only $O(\log n)$ vertices are reachable from any given vertex when $c < 1$, but that there is a unique "giant strong component" of size $\Omega(n)$ when $c > 1$.

The digraph that arises from random 2SAT isn't truly random, because its arcs come in pairs, $u \longrightarrow v$ and $\bar{v} \longrightarrow \bar{u}$. But intuitively we can expect that similar behavior will apply to digraphs that are just halfway random. For example, when the author generated a random 2SAT problem with $n = 1000000$ and $m = .99n$, the resulting digraph had only two complementary pairs of strong components with more than one vertex, and their sizes were only 2, 2 and 7, 7; so the clauses were easily satisfiable. Adding another $.01n$ clauses didn't increase the number of nontrivial strong components, and the problem remained satisfiable. But another experiment with $m = n = 1000000$ yielded a strong component of size 420, containing 210 variables and their complements; that problem was unsatisfiable.

Based on a similar intuition into the underlying structure, Chvátal and Reed introduced the following "snares and snakes" approach to the proof of Theorem C: Let's say that an *s-chain* is any sequence of s strictly distinct literals; thus there are $2^s n^{\underline{s}}$ possible s-chains. Every s-chain C corresponds to clauses

$$(\bar{l}_1 \vee l_2),\ (\bar{l}_2 \vee l_3),\ \ldots,\ (\bar{l}_{s-1} \vee l_s), \tag{85}$$

s-snare
first moment principle
t-snake

which in turn correspond to two paths

$$l_1 \longrightarrow l_2 \longrightarrow l_3 \longrightarrow \cdots \longrightarrow l_s \quad \text{and} \quad \bar{l}_s \longrightarrow \cdots \longrightarrow \bar{l}_3 \longrightarrow \bar{l}_2 \longrightarrow \bar{l}_1 \tag{86}$$

in the digraph. An *s-snare* $(C; t, u)$ consists of an s-chain C and two indices t and u, where $1 < t \le s$ and $1 \le |u| < s$; it specifies the clauses (85) together with

$$(l_t \vee l_1) \qquad \text{and} \qquad (\bar{l}_s \vee l_u) \text{ if } u > 0,\ (\bar{l}_s \vee \bar{l}_{-u}) \text{ if } u < 0, \tag{87}$$

representing $\bar{l}_t \longrightarrow l_1$ and either $l_s \longrightarrow l_{|u|}$ or $l_s \longrightarrow \bar{l}_{|u|}$. The number of possible s-snares is $2^{s+1}(s-1)^2 n^{\underline{s}}$. Their clauses are rarely all present when m is small.

Exercise 200 explains how to use these definitions to prove Theorem C in the case $c < 1$. First we show that *every unsatisfiable 2SAT formula contains all the clauses of at least one snare.* Then, if we define the binary random variable

$$X(C; t, u) \;=\; [\text{all clauses of } (C; t, u) \text{ are present}], \tag{88}$$

it isn't difficult to prove that the snares of every s-chain C are unlikely:

$$\operatorname{E} X(C; t, u) \;\le\; m^{s+1} / \bigl(2n(n-1)\bigr)^{s+1}. \tag{89}$$

Finally, letting X be the sum of $X(C; t, u)$ over all snares, we obtain

$$\operatorname{E} X = \sum \operatorname{E} X(C; t, u) \le \sum_{s \ge 0} 2^{s+1} s(s-1) n^s \Bigl(\frac{m}{2n(n-1)}\Bigr)^{s+1} = \frac{2}{n}\Bigl(\frac{m}{n-1-m}\Bigr)^3$$

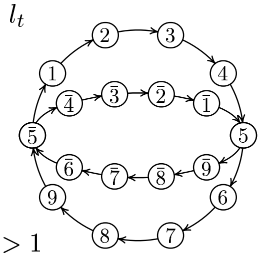
by Eq. 1.2.9–(20). This formula actually establishes a stronger form of (84), because it shows that $\operatorname{E} X$ is only $O(n^{-1/4})$ when $m = n - n^{3/4} > cn$. Thus

$$S_2\bigl(\lfloor n - n^{3/4} \rfloor, n\bigr) \;\ge\; \Pr(X = 0) \;=\; 1 - \Pr(X > 0) \;\ge\; 1 - O(n^{-1/4}) \tag{90}$$

by the first moment principle.

The other half of Theorem C can be proved by using the concept of a *t-snake*, which is the special case $(C; t, -t)$ of a $(2t-1)$-snare. In other words, given any chain $(l_1, \ldots, l_t, \ldots, l_{2t-1})$, with $s = 2t - 1$ and l_t in the middle, a t-snake generates the clauses (85) together with $(l_t \vee l_1)$ and $(\bar{l}_s \vee \bar{l}_t)$. When $t = 5$, for example, and $(l_1, \ldots, l_{2t-1}) = (x_1, \ldots, x_9)$, the $2t = 10$ clauses are

$$51,\ \bar{1}2,\ \bar{2}3,\ \bar{3}4,\ \bar{4}5,\ \bar{5}6,\ \bar{6}7,\ \bar{7}8,\ \bar{8}9,\ \bar{9}\bar{5},$$



and they correspond to 20 arcs that loop around to form a strong component as shown here. We will prove that, when $c > 1$ in (84), the digraph almost always contains such impediments to satisfiability.

Given a $(2t-1)$-chain C, where the parameter t will be chosen later, let

$$X_C \;=\; [\text{each clause of } (C; t, -t) \text{ occurs exactly once}]. \tag{91}$$

The expected value $\operatorname{E} X_C$ is clearly $f(2t)$, where

$$f(r) = m^{\underline{r}} \bigl(2n(n-1) - r\bigr)^{m-r} / \bigl(2n(n-1)\bigr)^m \tag{92}$$

is the probability that r specific clauses occur once each. Notice that

$$f(r) = \left(\frac{m}{2n(n-1)}\right)^r \left(1 + O\left(\frac{r^2}{m}\right) + O\left(\frac{rm}{n^2}\right)\right); \qquad (93)$$

thus the relative error will be $O(t^2/n)$ if $m = \Theta(n)$ as $n \to \infty$.

Now let $X = \sum X_C$, summed over all $R = 2^{2t-1} n^{\underline{2t-1}}$ possible t-snakes C; thus $\mathrm{E}\, X = Rf(2t)$. We want to show that $\Pr(X > 0)$ is very nearly 1, using the second moment principle; so we want to show that the expectation $\mathrm{E}\, X^2 = \mathrm{E}\left(\sum_C X_C\right)\left(\sum_D X_D\right) = \sum_C \sum_D \mathrm{E}\, X_C X_D$ is small. The key observation is that

$$\mathrm{E}\, X_C X_D = f(4t - r) \qquad \text{if } C \text{ and } D \text{ have exactly } r \text{ clauses in common.} \qquad (94)$$

Let p_r be the probability that a randomly chosen t-snake has exactly r clauses in common with the fixed snake $(x_1, \ldots, x_{2t-1})$. Then

$$\frac{\mathrm{E}\, X^2}{(\mathrm{E}\, X)^2} = \frac{R^2 \sum_{r=0}^{2t} p_r f(4t-r)}{R^2 f(2t)^2}$$
$$= \sum_{r=0}^{2t} p_r \frac{f(4t-r)}{f(2t)^2} = \sum_{r=0}^{2t} p_r \left(\frac{2n(n-1)}{m}\right)^r \left(1 + O\left(\frac{t^2}{n}\right)\right). \qquad (95)$$

By studying the interaction of snakes (see exercise 201) one can prove that

$$(2n)^r p_r \;=\; O(t^4/n) + O(t)[r \ge t] + O(n)[r = 2t], \qquad \text{for } 1 \le r \le 2t. \qquad (96)$$

Finally then, as explained in exercise 202, we can choose $t = \lfloor n^{1/5} \rfloor$ and $m = \lfloor n + n^{5/6} \rfloor$, to deduce a sharper form of (84) when $c > 1$:

$$S_2\big(\lfloor n + n^{5/6} \rfloor, n\big) \;=\; O(n^{-1/30}). \qquad (97)$$

(Deep breath.) Theorem C is proved. ▮

Much more precise results have been derived by B. Bollobás, C. Borgs, J. T. Chayes, J. H. Kim, and D. B. Wilson, in *Random Structures & Algorithms* **18** (2001), 201–256. For example, they showed that

$$S_2\big(\lfloor n - n^{3/4} \rfloor, n\big) = \exp\big(-\Theta(n^{-1/4})\big); \quad S_2\big(\lfloor n + n^{3/4} \rfloor, n\big) = \exp\big(-\Theta(n^{1/4})\big). \qquad (98)$$

Resolution. The backtracking process of Algorithms A, B, D, and L is closely connected to a logical proof procedure called *resolution*. Starting with a family of clauses called "axioms," there's a simple rule by which new clauses can be derived from this given set: Whenever both $x \vee A'$ and $\bar{x} \vee A''$ are in our repertoire of clauses, we're allowed to derive the "resolvent" clause $A = A' \vee A''$, denoted by $(x \vee A') \diamond (\bar{x} \vee A'')$. (See exercises 218 and 219.)

A *proof by resolution* consists of a directed acyclic graph (dag) whose vertices are labeled with clauses in the following way: (i) Every source vertex is labeled with an axiom. (ii) Every other vertex has in-degree 2. (iii) If the predecessors of vertex v are v' and v'', the label of v is $C(v) = C(v') \diamond C(v'')$.

When such a dag has a sink vertex labeled A, we call it a "resolution proof of A"; and if A is the empty clause, the dag is also called a "resolution refutation."

second moment principle
Bollobás
Borgs
Chayes
Kim
Wilson
resolution–
axioms
Notation $C' \diamond C''$
directed acyclic graph
dag
refutation

regular resolution
treelike resolution
Rivest
unnecessary branch
lookahead
Impagliazzo
Pudlák
Prover–Delayer game

The dag of a proof by resolution can be expanded to a binary tree, by replicating any vertex that has out-degree greater than 1. Such a tree is said to be *regular* if no path from the root to a leaf uses the same variable twice to form a resolvent. For example, Fig. 48 is a regular resolution tree that refutes Rivest's unsatisfiable axioms (6). All arcs in this tree are directed upwards.

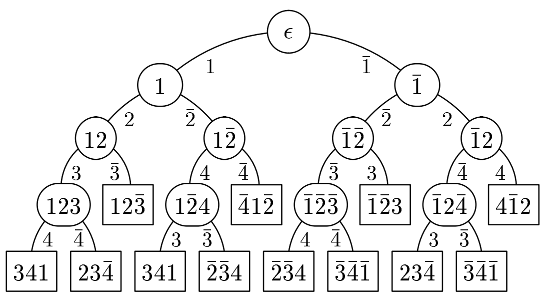


Fig. 48. One way to derive ϵ by resolving the inconsistent clauses (6).

Notice that Fig. 48 is essentially identical to Fig. 39 on page 33, the backtrack tree by which Algorithm D discovers that the clauses of (6) are unsatisfiable. In fact this similarity is no coincidence: *Every backtrack tree that records the behavior of Algorithm D on a set of unsatisfiable clauses corresponds to a regular resolution tree that refutes those axioms*, unless Algorithm D makes an unnecessary branch. (An unnecessary branch occurs if the algorithm tries $x \leftarrow 0$ and $x \leftarrow 1$ without using their consequences to discover an unsatisfiable subset of axioms.) Conversely, every regular refutation tree corresponds to a sequence of choices by which a backtrack-based SAT solver could prove unsatisfiability.

The reason behind this correspondence isn't hard to see. Suppose both values of x need to be tried in order to prove unsatisfiability. When we set $x \leftarrow 0$ in one branch of the backtrack tree, we replace the original clauses F by $F \mid \bar{x}$, as in (54). The key point is that *we can prove the empty clause by resolution from $F \mid \bar{x}$ if and only if we can prove x by resolution from F without resolving on x*. (See exercise 224.) Similarly, setting $x \leftarrow 1$ corresponds to changing the clauses from F to $F \mid x$.

Consequently, if F is an inconsistent set of clauses that has no short refutation tree, Algorithm D cannot conclude that those clauses are unsatisfiable unless it runs for a long time. Neither can Algorithm L, in spite of enhanced lookahead.

R. Impagliazzo and P. Pudlák [*SODA* **11** (2000), 128–136] have introduced an appealing *Prover–Delayer game*, with which it's relatively easy to demonstrate that certain sets of unsatisfiable clauses require large refutation trees. The Prover names a variable x, and the Delayer responds by saying either $x \leftarrow 0$ or $x \leftarrow 1$ or $x \leftarrow *$. In the latter case the Prover gets to decide the value of x; but the Delayer scores one point. The game ends when the current assignments have falsified at least one clause. If the Delayer has a strategy that guarantees a score of at least m points, exercise 226 shows that every refutation tree has at least 2^m

anti-maximal-element clauses
binary relation
irreflexive
transitive
partial ordering
maximal
Lauria
Stålmarck
hyperresolution

leaves; hence at least $2^m - 1$ resolutions must be done, and every backtrack-based solver needs $\Omega(2^m)$ operations to declare the clauses unsatisfiable.

We can apply their game, for example, to the following interesting clauses:

$$(\bar{x}_{jj}), \qquad \text{for } 1 \le j \le m; \tag{99}$$

$$(\bar{x}_{ij} \vee \bar{x}_{jk} \vee x_{ik}), \qquad \text{for } 1 \le i, j, k \le m; \tag{100}$$

$$(x_{j1} \vee x_{j2} \vee \cdots \vee x_{jm}), \qquad \text{for } 1 \le j \le m. \tag{101}$$

There are m^2 variables x_{jk}, for $1 \le j, k \le m$, which we can regard as the incidence matrix for a binary relation '$j \prec k$'. With this formulation, (99) says that the relation is irreflexive, and (100) says that it's transitive; thus, (99) and (100) amount to saying that $j \prec k$ is a *partial ordering*. Finally, (101) says that, for every j, there's a k with $j \prec k$. So these clauses state that there's a partial ordering on $\{1, \ldots, m\}$ in which no element is maximal; and they can't all be satisfied.

We can, however, always score $m - 1$ points if we're playing Delayer in that game, by using the following strategy suggested by Massimo Lauria: At every step we know an ordered set S of elements, regarded as "small"; initially $S = \emptyset$, and we'll have $S = \{j_1, \ldots, j_s\}$ when our score is s. Suppose the Prover queries x_{jk}, and $s < m-2$. If $j = k$, we naturally reply that $x_{jk} \leftarrow 0$. Otherwise, if $j \notin S$ and $k \notin S$, we respond $x_{jk} \leftarrow *$; then $s \leftarrow s+1$, and $j_s \leftarrow j$ or k according as the Prover specifies $x_{jk} \leftarrow 1$ or $x_{jk} \leftarrow 0$. Otherwise, if $j \in S$ and $k \notin S$, we respond $x_{jk} \leftarrow 1$; if $j \notin S$ and $k \in S$, we respond $x_{jk} \leftarrow 0$. Finally, if $j = j_a \in S$ and $k = j_b \in S$, we respond $x_{jk} \leftarrow [a < b]$. These responses always satisfy (99) and (100). And no clause of (101) becomes false before the Delayer is asked a question with $s = m - 2$. Then the response $x_{jk} \leftarrow *$ gains another point. We've proved

Theorem R. *Every refutation tree for the clauses* (99), (100), (101) *represents at least* $2^{m-1} - 1$ *resolution steps.* ▌

On the other hand, those clauses do have a refutation *dag* of size $O(m^3)$. Let I_j and T_{ijk} stand for the irreflexivity and transitivity axioms (99) and (100); and let $M_{jk} = x_{j1} \vee \cdots \vee x_{jk}$, so that (101) is M_{jm}. Then we have

$$M_{im} \diamond T_{imk} \;=\; M_{i(m-1)} \vee \bar{x}_{mk}, \qquad \text{for } 1 \le i, k < m. \tag{102}$$

Calling this new clause M'_{imk}, we can now derive

$$M_{j(m-1)} \;=\; \bigl(\bigl(\cdots\bigl((M_{mm} \diamond M'_{jm1}) \diamond M'_{jm2}\bigr) \diamond \cdots\bigr) \diamond M'_{jm(m-1)}\bigr) \diamond I_m,$$

for $1 \le j < m$. Hence $(m-1)^2 + (m-1)m$ resolutions have essentially reduced m to $m - 1$. Eventually we can therefore derive M_{11}; then $M_{11} \diamond I_1 = \epsilon$. [This elegant refutation is due to G. Stålmarck, *Acta Informatica* **33** (1996), 277–280.]

The method we've just used to obtain $M_{j(m-1)}$ from M_{mm} is, incidentally, a special case of a useful general formula called *hyperresolution* that is easily proved by induction on r:

$$\bigl(\cdots\bigl((C_0 \vee x_1 \vee \cdots \vee x_r) \diamond (C_1 \vee \bar{x}_1)\bigr) \diamond \cdots\bigr) \diamond (C_r \vee \bar{x}_r) = C_0 \vee C_1 \vee \cdots \vee C_r. \tag{103}$$

***Lower bounds for general resolution.** Let's change our perspective slightly: Instead of visualizing a proof by resolution as a directed graph, we can think of it as a "straight line" *resolution chain*, analogous to the addition chains of Section 4.6.3 and the Boolean chains of Section 7.1.2. A resolution chain based on m axioms $C_1, \ldots, C_m$ appends additional clauses $C_{m+1}, \ldots, C_{m+r}$, each of which is obtained by resolving two previous clauses of the chain. Formally, we have

$$C_i \;=\; C_{j(i)} \diamond C_{k(i)}, \qquad \text{for } m+1 \le i \le m+r, \tag{104}$$

where $1 \le j(i) < i$ and $1 \le k(i) < i$. It's a *refutation chain* for $C_1, \ldots, C_m$ if $C_{m+r} = \epsilon$. The tree in Fig. 48, for example, yields the refutation chain

$$12\bar{3}, 23\bar{4}, 341, 4\bar{1}2, \bar{1}23, \bar{2}34, \bar{3}\bar{4}\bar{1}, \bar{4}1\bar{2}, 123, 1\bar{2}4, \bar{1}2\bar{3}, \bar{1}2\bar{4}, 12, 1\bar{2}, \bar{1}\bar{2}, \bar{1}2, 1, \bar{1}, \epsilon$$

for the axioms (6); and there are many other ways to refute those axioms, such as

$$12\bar{3}, 23\bar{4}, 341, 4\bar{1}2, \bar{1}23, \bar{2}34, \bar{3}\bar{4}\bar{1}, \bar{4}1\bar{2}, 1\bar{2}\bar{3}, 1\bar{3}, 14, \bar{3}\bar{4}, 24, 2\bar{4}, 2, \bar{1}3, \bar{3}4, 1\bar{4}, \bar{3}, 1, \bar{1}, \epsilon. \tag{105}$$

This chain is quite different from Fig. 48, and perhaps nicer: It has three more steps, but after forming '$1\bar{2}\bar{3}$' it constructs only very short clauses.

We'll see in a moment that short clauses are crucial if we want short chains. That fact turns out to be important when we try to prove that certain easily understood families of axioms are inherently more difficult than (99), (100), and (101), in the sense that they can't be refuted with a chain of polynomial size.

Consider, for example, the well known "pigeonhole principle," which states that $m+1$ pigeons don't fit in m pigeon-sized holes. If x_{jk} means that pigeon j occupies hole k, for $0 \le j \le m$ and $1 \le k \le m$, the relevant unsatisfiable clauses are

$$(x_{j1} \vee x_{j2} \vee \cdots \vee x_{jm}), \qquad \text{for } 0 \le j \le m; \tag{106}$$

$$(\bar{x}_{ik} \vee \bar{x}_{jk}), \qquad \text{for } 0 \le i < j \le m \text{ and } 1 \le k \le m. \tag{107}$$

("Every pigeon has a hole, but no hole hosts more than one pigeon.") These clauses increased the pigeonhole principle's fame during the 1980s, when Armin Haken [*Theoretical Computer Science* **39** (1985), 297–308] proved that they have no short refutation chain. His result marked the first time that *any* set of clauses had been shown to be intractable for resolution in general.

It is absolutely necessary that two people have equally many hairs.
— JEAN APPIER HANZELET, *Recreation Mathematicque* (1624)

Haken's original proof was rather complicated. But simpler approaches were eventually found, culminating in a method by E. Ben-Sasson and A. Wigderson [*JACM* **48** (2001), 149–169], which is based on clause length and applies to many other sets of axioms. If α is any sequence of clauses, let us say that its *width*, written $w(\alpha)$, is the length of its longest clause or clauses. Furthermore, if $\alpha_0 = (C_1, \ldots, C_m)$, we write $w(\alpha_0 \vdash \epsilon)$ for the minimum of $w(\alpha)$ over all refutation chains $\alpha = (C_1, \ldots, C_{m+r})$ for α_0, and $\|\alpha_0 \vdash \epsilon\|$ for the minimum length r of all such chains. The following lemma is the key to proving lower bounds with Ben-Sasson and Wigderson's strategy:

resolution chain
refutation chain
pigeonhole principle
Haken
APPIER HANZELET
Ben-Sasson
Wigderson
width
notation $w(\alpha)$
notation $w(\alpha \vdash \epsilon)$
notation $\|\alpha \vdash C\|$

Lemma B. $\|\alpha_0 \vdash \epsilon\| \geq e^{(w(\alpha_0\vdash\epsilon)-1)^2/(8n)} - 2$, *for clauses in* $n \geq w(\alpha_0)^2$ *variables.* Thus there's exponential growth if we have $w(\alpha_0) = O(1)$ and $w(\alpha_0 \vdash \epsilon) = \Omega(n)$.

Proof. Let $\alpha = (C_1, \ldots, C_{m+r})$ be a refutation of α_0 with $r = \|\alpha_0 \vdash \epsilon\|$. We will say that a clause is "fat" if its length is W or more, where $W \geq w(\alpha_0)$ is a parameter to be set later. If $\alpha \setminus \alpha_0$ contains f fat clauses, those clauses contain at least Wf literals; hence some literal l appears in at least $Wf/(2n)$ of them.

Now $\alpha \mid l$, the chain obtained by replacing each clause C_j by $C_j \mid l$, is a refutation of $\alpha_0 \,|\, l$ that contains at most $\lfloor \rho f \rfloor$ fat clauses, where $\rho = 1 - W/(2n)$. (The clause $C_j \,|\, l$ will be $\wp$ if $l \in C_j$, thus tautological and effectively absent.)

Suppose $f < \rho^{-b}$ for some integer b. We will prove, by induction on b and secondarily on the total length of all clauses, that there's a refutation β of α_0 such that $w(\beta) \leq W + b$. This assertion holds when $b = 0$, since $W \geq w(\alpha_0)$. If $b > 0$, there's a refutation β_0 of $\alpha_0 \,|\, l$ with $w(\beta_0) \leq W + b - 1$, when we choose l as above, because $\rho f < \rho^{1-b}$ and $\alpha \,|\, l$ refutes $\alpha_0 \,|\, l$. Then we can form a resolution chain β_1 that derives $\bar l$ from α_0, by inserting $\bar l$ appropriately into clauses of β_0. And there's a simple chain β_2 that derives the clauses of $\alpha_0 \,|\, \bar l$ from α_0 and $\bar l$. There's also a refutation β_3 of $\alpha_0 \,|\, \bar l$ with $w(\beta_3) \leq W + b$, by induction, because $\alpha \,|\, \bar l$ refutes $\alpha_0 \,|\, \bar l$. Thus the combination $\beta = \{\beta_1, \beta_2, \beta_3\}$ refutes α_0, with

$$w(\beta) = \max(w(\beta_0)+1, w(\beta_2), w(\beta_3)) \leq \max(W+b, w(\alpha_0), W+b) = W+b.$$

Finally, exercise 238 chooses W so that we get the claimed bound. ▌

The pigeon axioms are too wide to be inserted directly into Lemma B. But Ben-Sasson and Wigderson observed that a simplified version of those axioms, involving only clauses of 5SAT, is already intractable.

Notice that we can regard the variable x_{jk} as indicating the presence of an edge between a_j and b_k in a bipartite graph on the vertices $A = \{a_0, \ldots, a_m\}$ and $B = \{b_1, \ldots, b_m\}$. Condition (106) says that each a_j has degree ≥ 1, while condition (107) says that each b_k has degree ≤ 1. There is, however, a bipartite graph G_0 on those vertices for which each a_j has degree ≤ 5 and such that the following strong "expansion" condition is satisfied:

$$\textit{Every subset } A' \subseteq A \textit{ with } |A'| \leq m/3000 \textit{ has } |\partial A'| \geq |A'| \textit{ in } G_0. \qquad (108)$$

Here $\partial A'$ denotes the bipartite boundary of A', namely the set of all b_k that have exactly one neighbor in A'.

Given such a graph G_0, whose existence is proved (nonconstructively) in exercise 240, we can formulate a *restricted pigeonhole principle*, by which the pigeonhole clauses are unsatisfiable if we also require $\bar x_{jk}$ whenever $a_j \not\!\!- b_k$ in G_0.

Let $\alpha(G_0)$ denote the resulting clauses, which are obtained when axioms (106) and (107) are conditioned on all such literals $\bar x_{jk}$. Then $w(\alpha(G_0)) \leq 5$, and at most $5m + 5$ unspecified variables x_{jk} remain. Lemma B tells us that all refutation chains for $\alpha(G_0)$ have length $\exp \Omega(m)$ if we can prove that they all have width $\Omega(m)$. Haken's theorem, which asserts that all refutation chains for (106) and (107) also have length $\exp \Omega(m)$, will follow, because any short refutation would yield a short refutation of $\alpha(G_0)$ after conditioning on the $\bar x_{jk}$.

fat
tautological
Ben-Sasson
Wigderson
5SAT
bipartite graph
expansion
boundary
restricted pigeonhole principle
Haken

notation $F \vdash C$
notation $\mu(C)$
subadditive law
3SAT
random 3SAT
Chvátal
Szemerédi
first order logic
Robinson
Gentzen
cut rule
sequents
Robinson
Tseytin
graph-based axioms
Slisenko

Thus the following result gives our story a happy ending:

Theorem B. *The restricted pigeonhole axioms $\alpha(G_0)$ have refutation width*

$$w(\alpha(G_0) \vdash \epsilon) \geq m/6000. \tag{109}$$

Proof. We can assign a complexity measure to every clause C by defining

$$\mu(C) = \min\{\,|A'| \bigm| A' \subseteq A \text{ and } \alpha(A') \vdash C\,\}. \tag{110}$$

Here $\alpha(A')$ is the set of "pigeon axioms" (106) for $a_j \in A'$, together with all of the "hole axioms" (107); and $\alpha(A') \vdash C$ means that clause C can be proved by resolution when starting with only those axioms. If C is one of the pigeon axioms, this definition makes $\mu(C) = 1$, because we can let $A' = \{a_j\}$. And if C is a hole axiom, clearly $\mu(C) = 0$. The subadditive law

$$\mu(C' \diamond C'') \;\leq\; \mu(C') + \mu(C'') \tag{111}$$

also holds, because a proof of $C' \diamond C''$ needs at most the axioms of $\alpha(A') \cup \alpha(A'')$ if C' follows from $\alpha(A')$ and C'' follows from $\alpha(A'')$.

We can assume that $m \geq 6000$. And we must have $\mu(\epsilon) > m/3000$, because of the strong expansion condition (108). (See exercise 241.) Therefore every refutation of $\alpha(G_0)$ must contain a clause C with $m/6000 \leq \mu(C) < m/3000$; indeed, the first clause C_j with $\mu(C_j) \geq m/6000$ will satisfy this condition, by (111).

Let A' be a set of vertices with $|A'| = \mu(C)$ and $\alpha(A') \vdash C$. Also let b_k be any element of $\partial A'$, with a_j its unique neighbor in A'. Since $|A' \setminus a_j| < \mu(C)$, there must be an assignment of variables that satisfies all axioms of $\alpha(A' \setminus a_j)$, but falsifies C and the pigeon axiom for j. That assignment puts no two pigeons into the same hole, and it places every pigeon of $A' \setminus a_j$.

Now suppose C contains no literal of the form $x_{j'k}$ or $\bar{x}_{j'k}$, for any $a_{j'} \in A$. Then we could set $x_{j'k} \leftarrow 0$ for all j', without falsifying any axiom of $\alpha(A' \setminus a_j)$; and we could then make the axioms of $\alpha(\{a_j\})$ true by setting $x_{jk} \leftarrow 1$. But that change to the assignment would leave C false, contradicting our assumption that $\alpha(A') \vdash C$. Thus C contains some $\pm x_{j'k}$ for each $b_k \in \partial A'$; and we must have $w(C) \geq |\partial A'| \geq m/6000$. ∎

A similar proof establishes a linear lower bound on the refutation width, hence an exponential lower bound on the refutation length, of almost all random 3SAT instances with n variables and $\lfloor \alpha n \rfloor$ clauses, for fixed α as $n \to \infty$ (see exercise 243), a theorem of V. Chvátal and E. Szemerédi [*JACM* **35** (1988), 759–768].

Historical notes: Proofs by resolution, in the more general setting of first order logic, were introduced by J. A. Robinson in *JACM* **12** (1965), 23–41. [They're also equivalent to G. Gentzen's "cut rule for sequents," *Mathematische Zeitschrift* **39** (1935), 176–210, III.1.2 1.] Inspired by Robinson's paper, Gregory Tseytin developed the first nontrivial techniques to prove lower bounds on the length of resolution proofs, based on unsatisfiable graph axioms that are considered in exercise 245. His lectures of 1966 were published in Volume 8 of the Steklov Mathematical Institute Seminars in Mathematics (1968); see A. O. Slisenko's English translation, *Studies in Constructive Mathematics and Mathematical Logic*, part 2 (1970), 115–125.

Tseytin pointed out that there's a simple way to get around the lower bounds he had proved for his graph-oriented problems, by allowing new kinds of proof steps: Given any set of axioms F, we can introduce a new variable z that doesn't appear anywhere in F, and add three new clauses $G = \{xz, yz, \bar{x}\bar{y}\bar{z}\}$; here x and y are arbitrary literals of F. It's clear that F is satisfiable if and only if $F \cup G$ is satisfiable, because G essentially says that $z = \text{NAND}(x, y)$. Adding new variables in this way is somewhat analogous to using lemmas when proving a theorem, or to introducing a memo cache in a computer program.

His method, which is called *extended resolution*, can be much faster than pure resolution. For example, it allows the pigeonhole clauses (106) and (107) to be refuted in only $O(m^4)$ steps (see exercise 237). It doesn't appear to help much with certain other classes of problems such as random 3SAT; but who knows?

SAT solving via resolution. The concept of resolution also suggests alternative ways to solve satisfiability problems. In the first place we can use it to eliminate variables: If F is any set of clauses on n variables, and if x is one of those variables, we can construct a set F' of clauses on the other $n-1$ variables in such a way that F is satisfiable if and only if F' is satisfiable. The idea is simply to resolve every clause of the form $x \vee A'$ with every clause of the form $\bar{x} \vee A''$, and then to discard those clauses.

For example, consider the following six clauses in four variables:

$$1234,\ 1\bar{2},\ \bar{1}\bar{2}\bar{3},\ \bar{1}3,\ 2\bar{3},\ 3\bar{4}. \qquad (112)$$

We can eliminate the variable x_4 by forming $1234 \diamond 3\bar{4} = 123$. Then we can eliminate x_3 by resolving 123 and $\bar{1}3$ with $\bar{1}\bar{2}\bar{3}$ and $2\bar{3}$:

$$123 \diamond \bar{1}\bar{2}\bar{3} = \wp, \quad 123 \diamond 2\bar{3} = 12, \quad \bar{1}3 \diamond \bar{1}\bar{2}\bar{3} = \bar{1}\bar{2}, \quad \bar{1}3 \diamond 2\bar{3} = \bar{1}2.$$

Now we're left with $\{12, 1\bar{2}, \bar{1}2, \bar{1}\bar{2}\}$, because the tautology $\wp$ goes away. Eliminating x_2 gives $\{1, \bar{1}\}$, and eliminating x_1 gives $\{\epsilon\}$; hence (112) is unsatisfiable.

This method, which was originally proposed for hand calculation by E. W. Samson and R. K. Mueller in 1955, works beautifully on small problems. But why is it valid? There are (at least) two good ways to understand the reason. First, it's easy to see that F' is satisfiable whenever F is satisfiable, because $C' \diamond C''$ is true whenever C' and C'' are both true. Conversely, if F' is satisfied by some setting of the other $n-1$ variables, that setting must either satisfy A' for all clauses of the form $x \vee A'$, or else it must satisfy A'' for all clauses of the form $\bar{x} \vee A''$. (Otherwise neither A' nor A'' would be satisfied, for some A' and some A'', and the clause $A' \vee A''$ in F' would be false.) Thus at least one of the settings $x \leftarrow 0$ or $x \leftarrow 1$ will satisfy F.

Another good way to understand variable elimination is to notice that it corresponds to the elimination of an existential quantifier (see exercise 248).

Suppose p clauses of F contain x and q clauses contain $\bar{x}$. Then the elimination of x will give us at most pq new clauses, in the worst case; so F' will have no more clauses than F did, whenever $pq \le p + q$, namely when $(p-1)(q-1) \le 1$. This condition clearly holds whenever $p = 0$ or $q = 0$; indeed, we called x a "pure literal" when such cases arose in Algorithm A. The condition also holds whenever $p = 1$ or $q = 1$, and even when $p = q = 2$.

Tseytin
NAND
variables, introducing new
auxiliary variables
memo cache
extended resolution
random 3SAT
3SAT
tautology
Samson
Mueller
existential quantifier
quantifiers
pure literal

subsumed
Cook
Method I
subsumed
heuristics
Method IA
Reckhow

Furthermore we don't always get pq new clauses. Some of the resolvents might turn out to be tautologous, as above; others might be subsumed by existing clauses. (The clause C is said to *subsume* another clause C' if $C \subseteq C'$, in the sense that every literal of C appears also in C'. In such cases we can safely discard C'.) And some of the resolvents might also subsume existing clauses.

Therefore repeated elimination of variables doesn't always cause the set of clauses to explode. In the worst case, however, it can be quite inefficient.

In January of 1972, Stephen Cook showed his students at the University of Toronto a rather different way to employ resolution in SAT-solving. His elegant procedure, which he called "Method I," essentially learns new clauses by doing resolution on demand:

Algorithm I (*Satisfiability by clause learning*). Given m nonempty clauses $C_1 \wedge \cdots \wedge C_m$ on n Boolean variables $x_1 \ldots x_n$, this algorithm either proves them unsatisfiable or finds strictly distinct literals $l_1 \ldots l_n$ that satisfy them all. In the process, new clauses may be generated by resolution (and m will then increase).

I1. [Initialize.] Set $d \leftarrow 0$.

I2. [Advance.] If $d = n$, terminate successfully (the literals $\{l_1, \ldots, l_d\}$ satisfy $\{C_1, \ldots, C_m\}$). Otherwise set $d \leftarrow d+1$, and let l_d be a literal strictly distinct from $l_1, \ldots, l_{d-1}$.

I3. [Find falsified C_i.] If none of $C_1, \ldots, C_m$ are falsified by $\{l_1, \ldots, l_d\}$, go back to I2. Otherwise let C_i be a falsified clause.

I4. [Find falsified C_j.] (At this point we have $\bar{l}_d \in C_i \subseteq \{\bar{l}_1, \ldots, \bar{l}_d\}$, but no clause is contained in $\{\bar{l}_1, \ldots, \bar{l}_{d-1}\}$.) Set $l_d \leftarrow \bar{l}_d$. If none of $C_1, \ldots, C_m$ are falsified by $\{l_1, \ldots, l_d\}$, go back to I2. Otherwise let $\bar{l}_d \in C_j \subseteq \{\bar{l}_1, \ldots, \bar{l}_d\}$.

I5. [Resolve.] Set $m \leftarrow m+1$, $C_m \leftarrow C_i \diamond C_j$. Terminate unsuccessfully if C_m is empty. Otherwise set $d \leftarrow \max\{t \mid \bar{l}_t \in C_m\}$, $i \leftarrow m$, and return to I4. ▮

In step I5 the new clause C_m cannot be subsumed by any previous clause C_k for $k < m$, because $C_i \diamond C_j \subseteq \{\bar{l}_1, \ldots, \bar{l}_{d-1}\}$. Therefore, in particular, no clause is generated twice, and the algorithm must terminate.

This description is intentionally vague when it uses the word "let" in steps I2, I3, and I4: *Any* available literal l_d can be selected in step I2, and *any* falsified clauses C_i and C_j can be selected in steps I3 and I4, without making the method fail. Thus Algorithm I really represents a family of algorithms, depending on what heuristics are used to make those selections.

For example, Cook proposed the following way ("Method IA") to select l_d in step I2: Choose a literal that occurs most frequently in the set of currently unsatisfied clauses that have the fewest unspecified literals. When applied to the six clauses (112), this rule would set $l_1 \leftarrow 3$ and $l_2 \leftarrow 2$ and $l_3 \leftarrow 1$; then step I3 would find $C_i = \bar{1}\bar{2}\bar{3}$ false. So step I4 would set $l_3 \leftarrow \bar{1}$ and find $C_j = 1\bar{2}$ false, and step I5 would learn $C_7 = \bar{2}\bar{3}$. (See exercise 249 for the sequel.)

Cook's main interest when introducing Algorithm I was to minimize the number of resolution steps; he wasn't particularly concerned with minimizing the running time. Subsequent experiments by R. A. Reckhow [Ph.D. thesis

(Univ. Toronto, 1976), 81–84] showed that, indeed, relatively short resolution refutations are found with this approach. Furthermore, exercise 251 demonstrates that Algorithm I can handle the anti-maximal-element clauses (99)–(101) in polynomial time; thus it trounces the exponential behavior exhibited by all backtrack-based algorithms for this problem (see Theorem R).

On the other hand, Algorithm I does tend to fill memory with a great many new clauses when it is applied to large problems, and there's no obvious way to deal with those clauses efficiently. Therefore Cook's method did not appear to be of practical importance, and it remained unpublished for more than forty years.

Conflict driven clause learning. Algorithm I demonstrates the fact that unsuccessful choices of literals can lead us to discover valuable new clauses, thereby increasing our knowledge about the characteristics of a problem. When that idea was rediscovered from another point of view in the 1990s, it proved to be revolutionary: Significant industrial instances of SAT with many thousands or even millions of variables suddenly became feasible for the first time.

The name CDCL solver is often given to these new methods, because they are based on "conflict driven clause learning" rather than on classical backtracking. A CDCL solver shares many concepts with the DPLL algorithms that we've already seen; yet it is sufficiently different that we can understand it best by developing the ideas from scratch. Instead of implicitly exploring a search tree such as Fig. 39, a CDCL solver is built on the notion of a *trail*, which is a sequence $L_0L_1 \ldots L_{F-1}$ of strictly distinct literals that do not falsify any clause. We can start with $F = 0$ (the empty trail). As computation proceeds, our task is to extend the current trail until $F = n$, thus solving the problem, or to prove that no solution exists, by essentially learning that the empty clause is true.

Suppose there's a clause c of the form $l \lor \bar{a}_1 \lor \cdots \lor \bar{a}_k$, where a_1 through a_k are in the trail but l isn't. Literals in the trail are tentatively assumed to be true, and c must be satisfied; so we're forced to make l true. In such cases we therefore append l to the current trail and say that c is its "reason." (This operation is equivalent to what we called "unit propagation" in previous algorithms; those algorithms effectively removed the literals $\bar{a}_1, \ldots, \bar{a}_k$ when they became false, thereby leaving l as a "unit" all by itself. But our new viewpoint keeps each clause c intact, and knows all of its literals.) A *conflict* occurs if the complementary literal $\bar{l}$ is already in the trail, because l can't be both true and false; but let's assume for now that no conflicts arise, so that l can legally be appended by setting $L_F \leftarrow l$ and $F \leftarrow F + 1$.

If no such forcing clause exists, and if $F < n$, we choose a new distinct literal in some heuristic way, and we append it to the current trail with a "reason" of Λ. Such literals are called *decisions*. They partition the trail into a sequence of decision *levels*, whose boundaries can be indicated by a sequence of indices with $0 = i_0 \le i_1 < i_2 < i_3 < \cdots$; literal L_t belongs to level d if and only if $i_d \le t < i_{d+1}$. Level 0, at the beginning of the trail, is special: It contains literals that are forced by clauses of length 1, if such clauses exist. Any such literals are unconditionally true. Every other level begins with exactly one decision.

anti-maximal-element clauses
Cook
Conflict driven clause learning
CDCL solver
DPLL
trail
reason
unit propagation
conflict
forcing clause
asserting clause, see forcing clause
decisions
Level 0

dependence of literals
learning
conflict clause
Resolving

Consider, for example, the problem *waerden*$(3,3;9)$ of (9). The first items placed on the trail might be

t	L_t	level	reason	
0	$\bar6$	1	Λ	(a decision)
1	$\bar9$	2	Λ	(a decision)
2	3	2	396	(rearrangement of the clause 369)
3	$\bar4$	3	Λ	(a decision)
4	5	3	546	(rearrangement of the clause 456)
5	8	3	846	(rearrangement of the clause 468)
6	2	3	246	
7	$\bar7$	3	$\bar7\bar5\bar3$	(rearrangement of the clause $\bar3\bar5\bar7$)
8	$\bar2$	3	$\bar2\bar5\bar8$	(a conflict!)

(113)

Three decisions were made, and they started levels at $i_1 = 0$, $i_2 = 1$, $i_3 = 3$. Several clauses have been rearranged; we'll soon see why. And propagations have led to a conflict, because both 2 and $\bar2$ have been forced. (We don't actually consider the final entry L_8 to be part of the trail, because it contradicts L_6.)

If the reason for l includes the literal $\bar l'$, we say "l depends directly on l'." And if there's a chain of one or more direct dependencies, from l to l_1 to $\cdots$ to $l_k = l'$, we say simply that "l depends on l'." For example, 5 depends directly on $\bar4$ and $\bar6$ in (113), and $\bar2$ depends directly on 5 and 8; hence $\bar2$ depends on $\bar6$.

Notice that a literal can depend only on literals that precede it in the trail. Furthermore, every literal l that's forced at level $d > 0$ depends directly on some *other* literal on that same level d; otherwise l would already have been forced at a previous level. Consequently l must necessarily depend on the dth decision.

The reason for reasons is that we need to deal with conflicts. We will see that every conflict allows us to construct a new clause c that must be true whenever the existing clauses are satisfiable, although c itself does not contain any existing clause. Therefore we can "learn" c by adding it to the existing clauses, and we can try again. This learning process can't go on forever, because only finitely many clauses are possible. Sooner or later we will therefore either find a solution or learn the empty clause. That will be nice, especially if it happens sooner.

A conflict clause c on decision level d has the form $\bar l \vee \bar a_1 \vee \cdots \vee \bar a_k$, where l and all the a's belong to the trail; furthermore l and at least one a_i belong to level d. We can assume that l is rightmost in the trail, of all the literals in c. Hence l cannot be the dth decision; and it has a reason, say $l \vee \bar a'_1 \vee \cdots \vee \bar a'_{k'}$. Resolving c with this reason gives the clause $c' = \bar a_1 \vee \cdots \vee \bar a_k \vee \bar a'_1 \vee \cdots \vee \bar a'_{k'}$, which includes at least one literal belonging to level d. If more than one such literal is present, then c' is itself a conflict clause; we can set $c \leftarrow c'$ and repeat the process. Eventually we are bound to obtain a new clause c' of the form $\bar l' \vee \bar b_1 \vee \cdots \vee \bar b_r$, where l' is on level d and where b_1 through b_r are on lower levels.

Such a c' is learnable, as desired, because it can't contain any existing clauses. (Every subclause of c', including c' itself, would otherwise have given us something to force at a lower level.) We can now *discard* levels $> d'$ of the trail, where d' is the maximum level of b_1 through b_r; and—this is the punch line—

we can append $\bar{l}'$ to the end of level d', with c' as its reason. The forcing process now resumes at level d', as if the learned clause had been present all along.

For example, after the conflict in (113), the initial conflict clause is $c = \bar{2}\bar{5}\bar{8}$, our shorthand notation for $\bar{x}_2 \vee \bar{x}_5 \vee \bar{x}_8$; and its rightmost complemented literal in the trail is 2, because 5 and 8 came earlier. So we resolve c with 246, the reason for 2, and get $c' = 4\bar{5}6\bar{8}$. This new clause contains complements of three literals from level 3, namely $\bar{4}$, 5, and 8; so it's still a conflict clause. We resolve it with the reason for 8 and get $c' = 4\bar{5}6$. Again c' is a conflict clause. But the result of resolving this conflict with the reason for 5 is $c' = 46$, a clause that is falsified by the literals currently on the trail but has only $\bar{4}$ at level 3. Good — we have learned '46': In every solution to *waerden*$(3, 3; 9)$, either x_4 or x_6 must be true.

Thus the sequel to (113) is

t	L_t	level	reason		
0	$\bar{6}$	1	Λ	(a decision)	(114)
1	4	1	46	(the newly learned clause)	

and the next step will be to begin a new level 2, because nothing more is forced.

Notice that the former level 2 has gone away. We've learned that there was no need to branch on the decision variable x_9, because $\bar{6}$ already forces 4. This improvement to the usual backtrack regimen is sometimes called "backjumping," because we've jumped back to a level that can be regarded as the root cause of the conflict that was just discovered.

Exercise 253 explores a possible continuation of (114); dear reader, please jump to it now. Incidentally, the clause '46' that we learned in this example involves the complements of former decisions $\bar{4}$ and $\bar{6}$; but exercise 255 shows that newly learned clauses might not contain any decision variables whatsoever.

The process of constructing the learned clause from a conflict is not as difficult as it may seem, because there's an efficient way to perform all of the necessary resolution steps. Suppose, as above, that the initial conflict clause is $\bar{l} \vee \bar{a}_1 \vee \cdots \vee \bar{a}_k$. Then we "stamp" each of the literals a_i with a unique number s; and we also insert $\bar{a}_i$ into an auxiliary array, which will eventually hold the literals $\bar{b}_1, \ldots, \bar{b}_r$, whenever a_i is a literal that received its value on a level d' with $0 < d' < d$. We stamp l too; and we count how many literals of level d have thereby been stamped. Then we repeatedly go back through the trail until coming to a literal L_t whose stamp equals s. If the counter is bigger than 1 at this point, and if the reason of L_t is $L_t \vee \bar{a}'_1 \vee \cdots \vee \bar{a}'_{k'}$, we look at each a'_i, stamping it and possibly putting it into the b array if it had not already been stamped with s. Eventually the count of unresolved literals will decrease to 1; the learned clause is then $\bar{L}_t \vee \bar{b}_1 \vee \cdots \vee \bar{b}_r$.

These new clauses might turn out to be quite large, even when we're solving a problem whose clauses were rather small to start with. For example, Table 3 gives a glimpse of typical behavior in a medium-size problem. It shows the beginning of the trail generated when a CDCL solver was applied to the 2779 clauses of *waerden*$(3, 10; 97)$, after about 10,000 clauses had been learned. (Recall that this problem tries to find a binary vector $x_1x_2 \ldots x_{97}$ that has no three equally

backtrack
backjumping
look-back, see backjumping
stamp
waerden

redundant
unit propagation
lazy data structures

Table 3

THE FIRST LEVELS OF A MODERATE-SIZE TRAIL

t	L_t	level	reason	t	L_t	level	reason	t	L_t	level	reason
0	$\overline{53}$	1	Λ	15	70	11	70 36 53	30	08	15	08 46 27
1	55	2	Λ	16	35	12	Λ	31	65	15	65 46 27
2	44	3	Λ	17	39	13	Λ	32	60	15	60 46 53
3	54	4	Λ	18	$\overline{37}$	14	Λ	33	$\overline{50}$	15	∗∗
4	43	5	Λ	19	38	14	38 37 36	34	64	15	64 50 36
5	30	6	Λ	20	47	14	47 37 27	35	22	15	22 50 36
6	34	7	Λ	21	17	14	17 37 27	36	24	15	24 50 37
7	45	8	Λ	22	32	14	32 37 27	37	42	15	42 50 46
8	40	9	Λ	23	69	14	69 37 53	38	48	15	48 50 46
9	$\overline{27}$	10	Λ	24	21	14	21 37 53	39	73	15	73 50 27
10	79	10	79 53 27	25	$\overline{46}$	15	Λ	40	04	15	04 50 27
11	01	10	01 27 53	26	28	15	28 46 37	41	63	15	63 50 37
12	$\overline{36}$	11	Λ	27	41	15	41 46 36	42	33	16	Λ
13	18	11	18 36 27	28	26	15	26 46 36	43	51	17	Λ
14	19	11	19 36 53	29	56	15	56 46 36	44	$\overline{57}$	18	Λ

(Here ∗∗ stands for the previously learned clause $\overline{50}\,\overline{26}\,\overline{47}\,\overline{35}\,\overline{41}\,\overline{32}\,\overline{38}\,\overline{44}\,27\,\overline{45}\,\overline{55}\,\overline{65}\,\overline{60}\,\overline{70}\overline{30}$.)

spaced 0s and no ten equally spaced 1s.) Level 18 in the table has just been launched with the decision $L_{44} = \overline{57}$; and that decision will trigger the setting of many more literals 15, 49, 61, 68, 77, 78, 87, $\overline{96}$, ..., eventually leading to a conflict when trying to set L_{67}. The conflict clause turns out to have length 22:

$$53\,27\,36\,\overline{70}\,\overline{35}\,37\,\overline{69}\,\overline{21}\,46\,\overline{28}\,\overline{56}\,\overline{65}\,\overline{60}\,50\,\overline{64}\,\overline{24}\,\overline{42}\,\overline{73}\,\overline{63}\,\overline{33}\,\overline{51}\,57\,. \tag{115}$$

(Its literals are shown here in order of the appearance of their complements in the trail.) When we see such a monster clause, we might well question whether we really want to "learn" such an obscure fact!

A closer look, however, reveals that many of the literals in (115) are redundant. For example, $\overline{70}$ can safely be deleted, because its reason is '70 36 53'; both 36 and 53 already appear in (115), hence (115) ⋄ (70 36 53) gets rid of $\overline{70}$. Indeed, more than half of the literals in this example are redundant, and (115) can be simplified to the much shorter and more memorable clause

$$53\,27\,36\,\overline{35}\,37\,46\,50\,\overline{33}\,\overline{51}\,57\,. \tag{116}$$

Exercise 257 explains how to discover such simplifications, which turn out to be quite important in practice. For example, the clauses learned while proving *waerden*(3, 10; 97) unsatisfiable had an average length of 19.9 before simplification, but only 11.2 after; simplification made the algorithm run about 33% faster.

Most of the computation time of a CDCL solver is devoted to unit propagation. Thus we need to know when the value of a literal has been forced by previous assignments, and we hope to know it quickly. The idea of "lazy data structures," used above in Algorithm D, works nicely for this purpose, in the presence of long clauses, provided that we extend it so that every clause now has *two*

watched literals instead of one. If we know that the first two literals of a clause are not false, then we needn't look at this clause until one of them becomes false, even though other literals in the clause might be repeatedly veering between transient states of true, false, and undefined. And when a watchee does become false, we'll try to swap it with a nonfalse partner that can be watched instead. Propagations or conflicts will arise only when all of the remaining literals are false.

Algorithm C below therefore represents clauses with the following data structures: A monolithic array called MEM is assumed to be large enough to hold all of the literals in all of the clauses, interspersed with control information. Each clause $c = l_0 \vee l_1 \vee \cdots \vee l_{k-1}$ with $k > 1$ is represented by its starting position in MEM, with $\texttt{MEM}[c+j] = l_j$ for $0 \le j < k$. Its two watched literals are l_0 and l_1, and its size k is stored in $\texttt{MEM}[c-1]$. Unit clauses, for which $k = 1$, are treated differently; they appear in level 0 of the trail, not in MEM.

A learned clause c can be distinguished from an initial clause because it has a relatively high number, with $\texttt{MINL} \le c < \texttt{MAXL}$. Initially MAXL is set equal to MINL, the smallest cell in MEM that is available for learned clauses; then MAXL grows as new clauses are added to the repertoire. The set of learned clauses is periodically culled, so that the less desirable ones don't clutter up memory and slow things down. Additional information about a learned clause c is kept in $\texttt{MEM}[c-4]$ and $\texttt{MEM}[c-5]$, to help with this recycling process (see below).

Individual literals x_k and $\bar{x}_k$, for $1 \le k \le n$, are represented internally by the numbers $2k$ and $2k+1$ as in (57) above. And each of these $2n$ literals l has a list pointer W_l, which begins a linked list of the clauses in which l is watched. We have $W_l = 0$ if there is no such clause; but if $W_l = c > 0$, the next link in this "watch list" is in $\texttt{MEM}[c-2]$ if $l = l_0$, in $\texttt{MEM}[c-3]$ if $l = l_1$. [See Armin Biere, *Journal on Satisfiability, Boolean Modeling and Comp.* **4** (2008), 75–97.]

For example, the first few cells of MEM might contain the following data when we are representing the clauses (9) of *waerden*$(3,3;9)$:

$i =$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	...
$\texttt{MEM}[i] =$	9	45	3	2	4	6	15	51	3	4	6	8	21	45	3	6	8	10	...

(Clause 3 is '123', clause 9 is '234', clause 15 is '345', . . . , clause 45 is '135', clause 51 is '246', . . . ; the watch lists for literals x_1, x_2, x_3, x_4 begin respectively at $W_2 = 3$, $W_4 = 3$, $W_6 = 9$, $W_8 = 15$.)

The other major data structures of Algorithm C are focused on variables, not clauses. Each variable x_k for $1 \le k \le n$ has six current attributes $\texttt{S}(k)$, $\texttt{VAL}(k)$, $\texttt{OVAL}(k)$, $\texttt{TLOC}(k)$, $\texttt{HLOC}(k)$, and $\texttt{ACT}(k)$, which interact as follows: $\texttt{S}(k)$ is the "stamp" that's used during clause formation. If neither x_k nor $\bar{x}_k$ appears in the current trail, then $\texttt{VAL}(k) = -1$, and we say that x_k and its two literals are "free." But if $L_t = l$ is a literal of the trail, belonging to level d, we have

$$\texttt{VAL}(|l|) = 2d + (l \,\&\, 1) \quad \text{and} \quad \texttt{TLOC}(|l|) = t, \qquad \text{where } |l| = l \gg 1, \qquad (117)$$

and we say that l is "true" and $\bar{l}$ is "false." Thus a given literal l is false if and only if $\texttt{VAL}(|l|)$ is nonnegative and $\texttt{VAL}(|l|) + l$ is odd. In most cases a watched literal is not false; but there are exceptions to this rule (see exercise 261).

MEM
Unit clauses
level 0
recycling
watch list
Biere
stamp
free literals and free variables
true literals
false literals
watched literal

The attributes `ACT(`k`)` and `HLOC(`k`)` tell the algorithm how to select the next decision variable. Each variable x_k has an *activity* score `ACT(`k`)`, which heuristically estimates its desirability for branching. All of the free variables, and possibly others, are kept in an array called `HEAP`, which is arranged so that

$$\texttt{ACT(HEAP[}j\texttt{])} \le \texttt{ACT(HEAP[}(j-1)\gg 1\texttt{])} \qquad \text{for } 0 < j < h \qquad (118)$$

when it contains h elements (see Section 5.2.3). Thus `HEAP[0]` will always be a free variable of maximum activity, if it is free; so it's the variable that will be chosen to govern the decision when the trail starts to acquire a new level.

Activity scores help the algorithm to focus on recent conflicts. Suppose, for example, that 100 conflicts have been resolved, hence 100 clauses have been learned. Suppose further that x_j or $\bar{x}_j$ was stamped while resolving the conflicts numbered 3, 47, 95, 99, and 100; but x_k or $\bar{x}_k$ was stamped during conflicts 41, 87, 94, 95, 96, and 97. We could express their recent activity by computing

$$\texttt{ACT(}j\texttt{)} = \rho^0+\rho^1+\rho^5+\rho^{53}+\rho^{97}, \quad \texttt{ACT(}k\texttt{)} = \rho^3+\rho^4+\rho^5+\rho^6+\rho^{13}+\rho^{59},$$

where ρ is a damping factor (say $\rho = .95$), because $100-100=0$, $100-99=1$, $100-95=5$, …, $100-41=59$. In this particular case j would be considered to be less active than k unless ρ is less than about .8744.

In order to update the activity scores according to this measure, we would have to do quite a bit of recomputation whenever a new conflict occurs: The new scores would require us to multiply all n of the old scores by ρ, then to increase the activity of every newly stamped variable by 1. But there's a much better way, namely to compute ρ^{-100} times the scores shown above:

$$\texttt{ACT(}j\texttt{)} = \rho^{-3}+\rho^{-47}+\rho^{-95}+\rho^{-99}+\rho^{-100}, \ \ \texttt{ACT(}k\texttt{)} = \rho^{-41}+\cdots+\rho^{-96}+\rho^{-97}.$$

These newly scaled scores, suggested by Niklas Eén, give us the same information about the relative activity of each variable; and they're updated easily, because we need to do only one addition per stamped variable when resolving conflicts.

The only problem is that the new scores can become really huge, because ρ^{-M} can cause floating point overflow after the number M of conflicts becomes large. The remedy is to *divide* them all by 10^{100}, say, whenever any variable gets a score that exceeds 10^{100}. The `HEAP` needn't change, since (118) still holds.

During the algorithm the variable `DEL` holds the current scaling factor ρ^{-M}, divided by 10^{100} each time all of the activities have been rescaled.

Finally, the parity of `OVAL(`k`)` is used to control the polarity of each new decision in step C6. Algorithm C starts by simply making each `OVAL(`k`)` odd, although other initialization schemes are possible. Afterwards it sets `OVAL(`k`)` $\leftarrow$ `VAL(`k`)` whenever x_k leaves the trail and becomes free, as recommended by D. Frost and R. Dechter [*AAAI Conf.* **12** (1994), 301–306] and independently by K. Pipatsrisawat and A. Darwiche [*LNCS* **4501** (2007), 294–299], because experience has shown that the recently forced polarities tend to remain good. This technique is called "sticking" or "progress saving" or "phase saving."

Algorithm C is based on the framework of a pioneering CDCL solver called Chaff, and on an early descendant of Chaff called MiniSAT that was developed by N. Eén and N. Sörensson [*LNCS* **2919** (2004), 502–518].

activity
`ACT(`k`)`
heuristic
heap
focus
Mathews, Edwin Lee (41)
damping factor
Eén
floating point overflow
rescaling
Frost
Dechter
AAAI: American Association for Artificial Intelli
Pipatsrisawat
Darwiche
polarities
sticking values
progress saving
phase saving
Chaff
MiniSAT
Eén
Sörensson

purging
flushing
breadth-first search
unit propagation
Backjump

Algorithm C (*Satisfiability by CDCL*). Given a set of clauses on n Boolean variables, this algorithm finds a solution $L_0L_1\ldots L_{n-1}$ if and only if the clauses are satisfiable, meanwhile discovering M new ones that are consequences of the originals. After discovering $M_{\rm p}$ new clauses, it will purge some of them from its memory and reset $M_{\rm p}$; after discovering $M_{\rm f}$ of them, it will flush part of its trail, reset $M_{\rm f}$, and start over. (Details of purging and flushing will be discussed later.)

C1. [Initialize.] Set $\texttt{VAL}(k)\leftarrow\texttt{OVAL}(k)\leftarrow\texttt{TLOC}(k)\leftarrow-1$, $\texttt{ACT}(k)\leftarrow\texttt{S}(k)\leftarrow0$, $R_{2k}\leftarrow R_{2k+1}\leftarrow\Lambda$, $\texttt{HLOC}(k)\leftarrow p_k-1$, and $\texttt{HEAP}[p_k-1]\leftarrow k$, for $1\le k\le n$, where $p_1\ldots p_n$ is a random permutation of $\{1,\ldots,n\}$. Then input the clauses into MEM and the watch lists, as described above. Put the distinct unit clauses into $L_0L_1\ldots L_{F-1}$; but terminate unsuccessfully if there are contradictory clauses (l) and $(\bar l)$. Set MINL and MAXL to the first available position in MEM. (See exercise 260.) Set $i_0\leftarrow d\leftarrow s\leftarrow M\leftarrow G\leftarrow0$, $h\leftarrow n$, $\texttt{DEL}\leftarrow1$.

C2. [Level complete?] (The trail $L_0\ldots L_{F-1}$ now contains all of the literals that are forced by $L_0\ldots L_{G-1}$.) Go to C5 if $G=F$.

C3. [Advance G.] Set $l\leftarrow L_G$ and $G\leftarrow G+1$. Then do step C4 for all c in the watch list of $\bar l$, unless that step detects a conflict and jumps to C7. If there is no conflict, return to C2. (See exercise 261.)

C4. [Does c force a unit?] Let $l_0l_1\ldots l_{k-1}$ be the literals of clause c, where $l_1=\bar l$. (Swap $l_0\leftrightarrow l_1$ if necessary.) If l_0 is true, do nothing. Otherwise look for a literal l_j with $1<j<k$ that is not false. If such a literal is found, move c to the watch list of l_j. But if l_2, ..., l_{k-1} are all false, jump to C7 if l_0 is also false. On the other hand if l_0 is free, make it true by setting $L_F\leftarrow l_0$, $\texttt{TLOC}(|l_0|)\leftarrow F$, $\texttt{VAL}(|l_0|)\leftarrow 2d+(l_0\mathbin{\&}1)$, $R_{l_0}\leftarrow c$, and $F\leftarrow F+1$.

C5. [New level?] If $F=n$, terminate successfully. Otherwise if $M\ge M_{\rm p}$, prepare to purge excess clauses (see below). Otherwise if $M\ge M_{\rm f}$, flush literals as explained below and return to C2. Otherwise set $d\leftarrow d+1$ and $i_d\leftarrow F$.

C6. [Make a decision.] Set $k\leftarrow\texttt{HEAP}[0]$ and delete k from the heap (see exercises 262 and 266). If $\texttt{VAL}(k)\ge0$, repeat this step. Otherwise set $l\leftarrow 2k+(\texttt{OVAL}(k)\mathbin{\&}1)$, $\texttt{VAL}(k)\leftarrow 2d+(\texttt{OVAL}(k)\mathbin{\&}1)$, $L_F\leftarrow l$, $\texttt{TLOC}(|l|)\leftarrow F$, $R_l\leftarrow\Lambda$, and $F\leftarrow F+1$. (At this point $F=G+1$.) Go to C3.

C7. [Resolve a conflict.] Terminate unsuccessfully if $d=0$. Otherwise use the conflict clause c to construct a new clause $\bar l'\vee\bar b_1\vee\cdots\vee\bar b_r$ as described above. Set $\texttt{ACT}(|l|)\leftarrow\texttt{ACT}(|l|)+\texttt{DEL}$ for all literals l stamped during this process; also set d' to the maximum level occupied by $\{b_1,\ldots,b_r\}$ in the trail. (See exercise 263. Increasing $\texttt{ACT}(|l|)$ may also change HEAP.)

C8. [Backjump.] While $F>i_{d'+1}$, do the following: Set $F\leftarrow F-1$, $l\leftarrow L_F$, $k\leftarrow|l|$, $\texttt{OVAL}(k)\leftarrow\texttt{VAL}(k)$, $\texttt{VAL}(k)\leftarrow-1$, $R_l\leftarrow\Lambda$; and if $\texttt{HLOC}(|l|)<0$ insert k into HEAP (see exercise 262). Then set $G\leftarrow F$ and $d\leftarrow d'$.

C9. [Learn.] If $d>0$, set $c\leftarrow\texttt{MAXL}$, store the new clause in MEM at position c, and advance MAXL to the next available position in MEM. (Exercise 263 gives full details.) Set $M\leftarrow M+1$, $L_F\leftarrow l'$, $\texttt{TLOC}(|l'|)\leftarrow F$, $R_{l'}\leftarrow c$, $F\leftarrow F+1$, $\texttt{DEL}\leftarrow\texttt{DEL}/\rho$, and return to C3. ▮

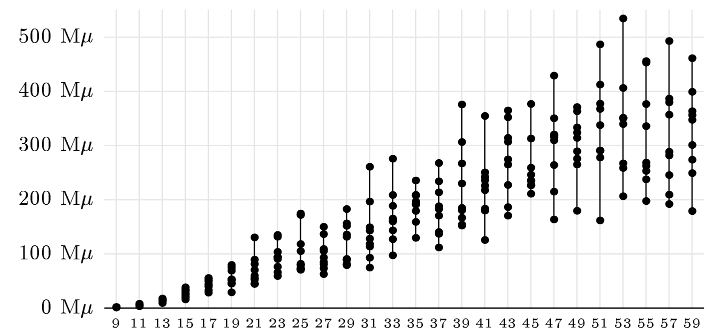
The high-level operations on data structures in this algorithm are spelled out in terms of elementary low-level steps in exercises 260–263. Exercises 266–271 discuss simple enhancements that were made in the experiments reported below.

decisions
nodes
three-coloring problems
flower snarks
snark graphs
fsnark
Certificates of unsatisfiability

Reality check: Although detailed statistics about the performance of Algorithm C on a wide variety of problems will be presented later, a few examples of typical behavior will help now to clarify how the method actually works in practice. Random choices make the running time of this algorithm more variable than it was in Algorithms A, B, D, or L; sometimes we're lucky, sometimes we're not.

In the case of *waerden*$(3, 10; 97)$, the modest 97-variable-and-2779-clause problem that was considered in Table 3, nine test runs of Algorithm C established unsatisfiability after making between 250 and 300 million memory accesses; the median was 272 Mμ. (This is more than twice as fast as our best previous time, which was obtained with Algorithm L.) The average number of decisions made — namely the number of times $L_F \leftarrow l$ was done in step C6 — was about 63 thousand; this compares to 1701 "nodes" in Algorithm L, step L3, and 100 million nodes in Algorithms A, B, D. About 53 thousand clauses were learned, having an average size of 11.5 literals (after averaging about 19.9 before simplification).

Fig. 49. It is not possible to color the edges of the flower snark graph J_q with three colors, when q is odd. Algorithm C is able to prove this with amazing speed: Computation times (in megamems) are shown for nine trials at each value of q.



Algorithm C often speeds things up much more dramatically, in fact. For example, Fig. 49 shows how it whips through a sequence of three-coloring problems that are based on "flower snarks." Exercise 176 defines *fsnark*(q), an interesting set of $42q + 3$ unsatisfiable clauses on $18q$ variables. The running time of Algorithms A, B, D, and L on *fsnark*(q) is proportional to 2^q, so it's way off the chart — well over a gigamem already when $q = 19$. But Algorithm C polishes off the case $q = 99$ in that same amount of time (thus winning by 24 orders of magnitude)! On the other hand, no satisfactory theoretical explanation for the apparently linear behavior in Fig. 49 is presently known.

Certificates of unsatisfiability. When a SAT solver reports that a given instance is satisfiable, it also produces a set of distinct literals from which we can easily check that every clause is satisfied. But if its report is *negative* — UNSAT — how confident can we be that such a claim is true? Maybe the implementation contains a subtle error; after all, large and complicated programs are notoriously buggy, and computer hardware isn't perfect either. A negative answer can therefore leave both programmers and users unsatisfied, as well as the problem.

We've seen that unsatisfiability can be proved rigorously by constructing a resolution refutation, namely a chain of resolution steps that ends with the empty clause ϵ, as in Fig. 48. But such refutations amount to the construction of a huge directed acyclic graph.

A much more compact characterization of unsatisfiability is possible. Let's say that the sequence of clauses $(C_1, C_2, \ldots, C_t)$ is a *certificate of unsatisfiability* for a family of clauses F if $C_t = \epsilon$, and if we have

$$F \wedge C_1 \wedge \cdots \wedge C_{i-1} \wedge \overline{C}_i \;\vdash_1\; \epsilon \qquad \text{for } 1 \le i \le t. \tag{119}$$

Here the subscript 1 in '$G \vdash_1 \epsilon$' means that the clauses G lead to a contradiction by *unit propagation*; and if C_i is the clause $(a_1 \vee \cdots \vee a_k)$, then $\overline{C}_i$ is an abbreviation for the conjunction of unit clauses $(\bar{a}_1) \wedge \cdots \wedge (\bar{a}_k)$.

For example, let $F = R$ be Rivest's clauses (6), which were proved unsatisfiable in Fig. 48. Then $(12, 1, 2, \epsilon)$ is a certificate of unsatisfiability, because

$$\begin{array}{ll}
R \wedge \bar{1} \wedge \bar{2} \vdash_1 \bar{3} \vdash_1 \bar{4} \vdash_1 \epsilon & (\text{using } 12\bar{3},\ 23\bar{4},\ \text{and } 341);\\
R \wedge 12 \wedge \bar{1} \vdash_1 2 \vdash_1 \bar{4} \vdash_1 \bar{3} \vdash_1 \epsilon & (\text{using } 12,\ \bar{4}1\bar{2},\ \bar{2}\bar{3}4,\ \text{and } 341);\\
R \wedge 12 \wedge 1 \wedge \bar{2} \vdash_1 4 \vdash_1 3 \vdash_1 \epsilon & (\text{using } 4\bar{1}2,\ 23\bar{4},\ \text{and } \bar{3}\bar{4}\bar{1});\\
R \wedge 12 \wedge 1 \wedge 2 \vdash_1 3 \vdash_1 4 \vdash_1 \epsilon & (\text{using } \bar{1}\bar{2}3,\ \bar{2}\bar{3}4,\ \text{and } \bar{3}\bar{4}\bar{1}).
\end{array}$$

A certificate of unsatisfiability gives a convincing proof, since (119) implies that each C_i must be true whenever F, C_1, $\ldots$, C_{i-1} are true. And it's easy to check whether or not $G \vdash_1 \epsilon$, for any given set of clauses G, because everything is forced and no choices are involved. Unit propagation is analogous to water flowing downhill; we can be pretty sure that it has been implemented correctly, even if we don't trust the CDCL solver that generated the certificate being checked.

E. Goldberg and Y. Novikov [*Proceedings of DATE: Design, Automation and Test in Europe* **6**,1 (2003), 886–891] have pointed out that CDCL solvers actually produce such certificates as a natural byproduct of their operation:

Theorem G. *If Algorithm C terminates unsuccessfully, the sequence* $(C_1, C_2, \ldots, C_t)$ *of clauses that it has learned is a certificate of unsatisfiability.*

Proof. It suffices to show that, whenever Algorithm C has learned the clause $C' = \bar{l}' \vee \bar{b}_1 \vee \cdots \vee \bar{b}_r$, unit propagation will deduce ϵ if we append the unit clauses $(l') \wedge (b_1) \wedge \cdots \wedge (b_r)$ to the clauses that the algorithm already knows. The key point is that C' has essentially been obtained by repeated resolution steps,

$$C' = \bigl(\ldots((C \diamond R_{l_1}) \diamond R_{l_2}) \diamond \cdots\bigr) \diamond R_{l_s}, \tag{120}$$

where C is the original conflict clause and R_{l_1}, R_{l_2}, $\ldots$, R_{l_s} are the reasons for each literal that was removed while C' was constructed in step C7. More precisely, we have $C = A_0$ and $R_{l_i} = l_i \vee A_i$, where all literals of $A_0 \cup A_1 \cup \cdots \cup A_s$ are false (their complements appear in the trail); and

$$\begin{aligned}
&\bar{l}_i \in A_0 \cup \cdots \cup A_{i-1}, \text{ for } 1 \le i \le s;\\
&A_0 \cup A_1 \cup \cdots \cup A_s = \{\bar{l}', \bar{l}_1, \ldots, \bar{l}_s, \bar{b}_1, \ldots, \bar{b}_r\}.
\end{aligned} \tag{121}$$

Thus the known clauses, plus b_1, $\ldots$, b_r, and l', will force l_s using clause R_{l_s}. And l_{s-1} will then be forced, using $R_{l_{s-1}}$. And so on. ▌

resolution refutation
clausal proofs, see certificates of unsat
notation $F \vdash_1 \epsilon$
unit propagation
unit clauses
Rivest
Goldberg
Novikov
learned clauses, sequence of
resolution
conflict clause

reverse unit propagation
Van Gelder
purges
Tseytin
extended resolution
autarkies
Wetzler
Heule
Hunt
Purging unhelpful clauses

Since the unit literals in this proof are propagated in reverse order l_s, l_{s-1}, ..., l_1 from the resolution steps in (120), this certificate-checking procedure has become known as "reverse unit propagation" [see A. Van Gelder, *Proc. Int. Symp. on Artificial Intelligence and Math.* **10** (2008), 9 pages, online as `ISAIM2008`].

Notice that the proof of Theorem G doesn't claim that reverse unit propagation will reconstruct the precise reasoning by which Algorithm C learned a clause. Many different downhill paths to ϵ, built from $\vdash_1$ steps, usually exist in a typical situation. We merely have shown that every clause learnable from a single conflict does imply the existence of at least one such downhill path.

Many of the clauses learned during a typical run of Algorithm C will be "shots in the dark," which turn out to have been aimed in unfruitful directions. Thus the certificates in Theorem G will usually be longer than actually necessary to demonstrate unsatisfiability. For example, Algorithm C learns about 53,000 clauses when refuting $waerden(3,10;97)$, and about 135,000 when refuting $fsnark(99)$; but fewer than 50,000 of the former, and fewer than 47,000 of the latter, were actually used in subsequent steps. Exercise 284 explains how to shorten a certificate of unsatisfiability while checking its validity.

An unexpected difficulty arises, however: We might spend more time verifying a certificate than we needed to generate it! For example, a certificate for $waerden(3,10;97)$ was found in 272 megamems, but the time needed to check it with straightforward unit-propagations was actually 2.2 *giga*mems. Indeed, this discrepancy becomes significantly worse in larger problems, because a simple program for checking must keep all of the clauses active in its memory. If there are a million active clauses, there are two million literals being watched; hence every change to a literal will require many updates to the data structures.

The solution to this problem is to provide extra hints to the certificate checker. As we are about to see, Algorithm C does *not* keep all of the learned clauses in its memory; it systematically purges its collection, so that the total number stays reasonable. At such times it can also inform the certificate checker that the purged clauses will no longer be relevant to the proof.

Further improvements also allow annotated certificates to accommodate stronger proof rules, such as Tseytin's extended resolution and techniques based on generalized autarkies; see N. Wetzler, M. J. H. Heule, and W. A. Hunt, Jr., *LNCS* **8561** (2014), 422–429.

Whenever a family of clauses has a certificate of unsatisfiability, a variant of Algorithm C will actually find one that isn't too much longer. (See exercise 386.)

***Purging unhelpful clauses.** After thousands of conflicts have occurred, Algorithm C has learned thousands of new clauses. New clauses guide the search by steering us away from unproductive paths; but they also slow down the propagation process, because we have to watch them.

We've seen that certificates can usually be shortened; therefore we know that many of the learned clauses will probably never be needed again. For this reason Algorithm C periodically attempts to weed out the ones that appear to be more harmful than helpful, by ranking the clauses that have accumulated.

HOLMES
Doyle
waerden(3, 10; 97)
author
discarding
reason
trail
heuristics
Audemard
Simon
literal block distance
signature
literal block distance
glucose, see literal block distance

I consider that a man's brain originally is like a little empty attic, and you have to stock it with such furniture as you choose. . . . the skilled workman is very careful indeed as to what he takes into his brain-attic. . . . It is a mistake to think that that little room has elastic walls and can distend to any extent. . . . It is of the highest importance, therefore, not to have useless facts elbowing out the useful ones.

— SHERLOCK HOLMES, in *A Study in Scarlet* (1887)

Algorithm C initiates a special clause-refinement process as soon as it has learned $M \geq M_{\rm p}$ clauses and arrived at a reasonably stable state (step C5). Let's continue our running example, *waerden*(3, 10; 97), in order to make the issues concrete. If $M_{\rm p}$ is so huge that no clauses are ever thrown away, a typical run will learn roughly 48 thousand clauses, and do roughly 800 megamems of computation, before proving unsatisfiability. But if $M_{\rm p} = 10000$, it will learn roughly 50 thousand clauses, and the computation time will go down to about 500 megamems. In the latter case the total number of learned clauses in memory will rarely exceed 10 thousand.

Indeed, let's set $M_{\rm p} = 10000$ and take a close look at exactly what happened during the author's first experiments. Algorithm C paused to reconnoiter the situation after having learned 10002 clauses. At that point only 6252 of those 10002 clauses were actually present in memory, however, because of the clause-discarding mechanism discussed in exercise 271. Some clauses had length 2, while the maximum size was 24 and the median was 11; here's a complete histogram:

2 9 49 126 216 371 542 719 882 1094 661 540 414 269 176 111 35 20 10 3 1 1 1.

Short clauses tend to be more useful, because they reduce more quickly to units.

A learned clause cannot be purged if it is the reason for one of the literals on the trail. In our example, 12 of the 6252 fell into this category; for instance, $\overline{30}$ appeared on level 10 of the trail because '$\overline{30}\ \overline{33}\ \overline{39}\ 41\ \overline{42}\ \overline{45}\ 46\ \overline{48}\ \overline{54}\ \overline{57}$' had been learned, and we may need to know that clause in a future resolution step.

The purging process will try to remove at least half of the existing learned clauses, so that at most 3126 remain. We aren't allowed to touch the 12 reason-bound ones; hence we want to forget 3114 of the other 6240. Which of them should we expel?

Among many heuristics that have been tried, the most successful in practice are based on what Gilles Audemard and Laurent Simon have called "literal block distance" [see *Proc. Int. Joint Conference on Artificial Intelligence* **21** (2009), 399–404]. They observed that each level of the trail can be considered to be a block of more-or-less related variables; hence a long clause might turn out to be more useful than a short clause, if the literals of the long one all lie on just one or two levels while the literals of the short one belong to three or more.

Suppose all the literals of a clause $C = l_1 \vee \cdots \vee l_r$ appear in the trail, either positively as l_j or negatively as $\bar{l}_j$. We can group them by level so that exactly $p+q$ levels are represented, where p of the levels contain at least one positive l_j and the other q contain nothing but $\bar{l}_j$'s. Then (p,q) is the *signature* of C with respect to the trail, and $p+q$ is the literal block distance. For example, the very

first clause learned from *waerden*$(3,10;97)$ in the author's test run was

VAL
Goultiaeva
Bacchus
full run

$$\overline{11}\ \overline{16}\ \overline{21}\ \overline{26}\ \overline{36}\ \overline{46}\ \overline{51}\ 61\ 66\ 91; \qquad (122)$$

later, when it was time to rank clauses for purging, the values and trail levels of those literals were specified by `VAL(11)`, `VAL(16)`, ..., `VAL(91)`, which were

$$20\ 21\ 21\ 21\ 20\ 15\ 16\ 8\ 14\ 20.$$

Thus 61 was true on level $8 \gg 1 = 4$; $\overline{46}$ and 66 were true on level $15 \gg 1 = 14 \gg 1 = 7$; $\overline{51}$ was false on level 8; the others were a mixture of true and false on level 10; hence (122) had $p = 3$ and $q = 1$ with respect to the current trail.

If C has signature (p,q) and C' has signature (p',q'), where $p \le p'$ and $q \le q'$ and $(p,q) \ne (p',q')$, we can expect that C is more likely than C' to be useful in future propagations. The same conclusion is plausible also when $p+q = p'+q'$ and $p < p'$, because C' won't force anything until literals from at least $p+1$ different levels change sign. These intuitive expectations are borne out by the following detailed data obtained from *waerden*$(3,10;97)$:

$$\begin{pmatrix} 0 & 4 & 17 & 22 & 30 & 54 & 67 & 99 & 17 \\ 17 & 81 & 191 & 395 & 360 & 404 & 438 & 66 & 6 \\ 63 & 232 & 463 & 536 & 521 & 386 & 117 & 6 & 0 \\ 52 & 243 & 291 & 298 & 308 & 112 & 22 & 0 & 0 \\ 18 & 59 & 86 & 77 & 53 & 7 & 0 & 0 & 0 \\ 0 & 8 & 3 & 10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 9 & 15 & 21 & 16 & 15 & 3 & 0 \\ 7 & 26 & 74 & 107 & 82 & 57 & 16 & 1 & 0 \\ 20 & 74 & 104 & 86 & 61 & 21 & 9 & 0 & 0 \\ 13 & 40 & 37 & 16 & 14 & 4 & 0 & 0 & 0 \\ 6 & 10 & 9 & 4 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

The matrix on the left shows how many of the 6240 eligible clauses had a given signature (p,q), for $1 \le p \le 7$ and $0 \le q \le 8$; the matrix on the right shows how many would have been used to resolve future conflicts, if none of them had been removed. There were, for example, 536 learned clauses with $p = q = 3$, of which only 86 actually turned out to be useful. This data is illustrated graphically in Fig. 50, which shows gray rectangles whose areas correspond to the left matrix, overlaid by black rectangles whose areas correspond to the right matrix. We can't predict the future, but small (p,q) tends to increase the ratio of black to gray.

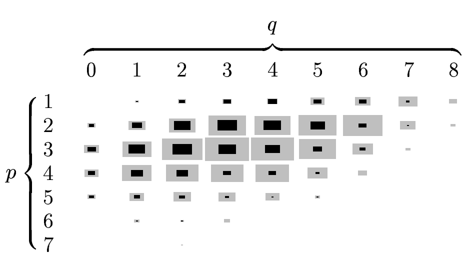


Fig. 50. Learned clauses that have p positive and q all-negative levels. The gray ones will never be used again. Unfortunately, there's no easy way to distinguish gray from black without being clairvoyant.

An alert reader will be wondering, however, how such signatures were found, because we can't compute them for all clauses until all variables appear in the trail — and that doesn't happen until all clauses are satisfied! The answer [see A. Goultiaeva and F. Bacchus, *LNCS* **7317** (2012), 30–43] is that it's quite possible to carry out a "full run" in which *every* variable is assigned a value, by making only a slight change to the normal behavior of Algorithm C: Instead

backjumping
VAL
OVAL
author
tie-breakers
literal block distance
ACT(c)
clause activity
activity score

of resolving conflicts immediately and backjumping, we can carry on after each conflict until all propagations cease, and we can continue to build the trail in the same way until every variable is present on some level. Conflicts may have occurred on several different levels; but we can safely resolve them later, learning new clauses at that time. Meanwhile, a full trail allows us to compute signatures based on `VAL` fields. And those `VAL` fields go into the `OVAL` fields after backjumping, so the variables in each block will tend to maintain their relationships.

The author's implementation of Algorithm C assigns an eight-bit value

$$\texttt{RANGE}(c) \;\leftarrow\; \min\bigl(\lfloor 16(p+\alpha q)\rfloor, 255\bigr) \qquad (123)$$

to each clause c; here α is a parameter, $0 \le \alpha \le 1$. We also set $\texttt{RANGE}(c) \leftarrow 0$ if c is the reason for some literal in the trail; $\texttt{RANGE}(c) \leftarrow 256$ if c is satisfied at level 0. If there are m_j clauses of range j, and if we want to keep at most T clauses in memory, we find the largest $j \le 256$ such that

$$m_j > 0 \quad \text{and} \quad s_j \;=\; m_0 + m_1 + \cdots + m_{j-1} \;\le\; T. \qquad (124)$$

Then we retain all clauses for which $\texttt{RANGE}(c) < j$, together with $T - s_j$ "tie-breakers" that have $\texttt{RANGE}(c) = j$ (unless $j = 256$). When α has the relatively high value $\frac{15}{16} = .9375$, this rule essentially preserves as many clauses of small literal block distance as it can; and for constant $p+q$ it favors those with small p.

For example, with $\alpha = \frac{15}{16}$ and the data from Fig. 50, we save clauses that have $p = (1,2,3,4,5)$ when $q \le (5,4,3,2,0)$, respectively. This gives us $s_{95} = 12 + 3069$ clauses, just 45 shy of our target $T = 3126$. So we also choose 45 tie-breakers from among the 59 clauses that have $\texttt{RANGE}(c) = 95$, $(p,q) = (5,1)$.

Tie-breaking can be done by using a secondary heuristic $\texttt{ACT}(c)$, "clause activity," which is analogous to the activity score of a variable but it is more easily maintained. If clause c has been used to resolve the conflicts numbered 3, 47, 95, 99, and 100, say, then

$$\texttt{ACT}(c) \;=\; \varrho^{-3} + \varrho^{-47} + \varrho^{-95} + \varrho^{-99} + \varrho^{-100}. \qquad (125)$$

This damping factor ϱ (normally .999) is independent of the factor ρ that is used for variable activities. In the case of Fig. 50, if the 59 clauses with $(p,q) = (5,1)$ are arranged in order of increasing `ACT` scores, the gray-and-black pattern is

So if we retain the 45 with highest activity, we pick up 8 of the 10 that turn out to be useful. (Clause activities are imperfect predictors, but they are usually somewhat better than this example implies.)

Exercises 287 and 288 present full details of clause purging in accordance with these ideas. One question remains: After we've completed a purge, when should we schedule the next one? Successful results are obtained by having two parameters, $\Delta_{\rm p}$ and $\delta_{\rm p}$. Initially $M_{\rm p} = \Delta_{\rm p}$; then after each purge, we set $\Delta_{\rm p} \leftarrow \Delta_{\rm p} + \delta_{\rm p}$ and $M_{\rm p} \leftarrow M_{\rm p} + \Delta_{\rm p}$. For example, if $\Delta_{\rm p} = 10000$ and $\delta_{\rm p} = 100$, purging will occur after approximately 10000, 20100, 30300, 40600, $\ldots$, $k\Delta_{\rm p} + \binom{k}{2}\delta_{\rm p}$,

... clauses have been learned; and the number of clauses at the beginning of the kth round will be approximately $20000+200k = 2\Delta_\mathrm{p} + 2k\delta_\mathrm{p}$. (See exercise 289.)

We've based this discussion on *waerden*(3, 10; 97), which is quite a simple problem. Algorithm C's gain from clause-purging on larger problems is naturally much more substantial. For example, *waerden*(3, 13; 160) is only a bit larger than *waerden*(3, 10; 97). With $\Delta_\mathrm{p} = 10000$ and $\delta_\mathrm{p} = 100$, it finishes in 132 gigamems, after learning 9.5 million clauses and occupying only 503 thousand MEM cells. Without purging, it proves unsatisfiability after learning only 7.1 million clauses, yet at well over ten times the cost: 4307 gigamems, and 102 million cells of MEM.

***Flushing literals and restarting.** Algorithm C interrupts itself in step C5 not only to purge clauses but also to "flush literals" that may not have been the best choices for decisions in the trail. The task of solving a tough satisfiability problem is a delicate balancing act: We don't want to get bogged down in the wrong part of the search space; but we also don't want to lose the fruits of hard work by "throwing out the baby with the bath water." A nice compromise has been found by Peter van der Tak, Antonio Ramos, and Marijn Heule [*J. Satisfiability, Bool. Modeling and Comp.* **7** (2011), 133–138], who devised a useful way to rejuvenate the trail periodically by following trends in the activity scores ACT(k).

Let's go back to Table 3, to illustrate their method. After learning the clause (116), Algorithm C will update the trail by setting $L_{44} \leftarrow 57$ on level 17; that will force $L_{45} \leftarrow \overline{66}$, because 39, 42, ..., 63 have all become true; and further positive literals 6, 58, 82, 86, 95, 96 will also join the trail in some order. Step C5 might then intervene to suggest that we should contemplate flushing some or all of the $F = 52$ literals whose values are currently assigned.

The decision literals $\overline{53}$, 55, 44, ..., 51 on levels 1, 2, 3, ..., 17 each were selected because they had the greatest current activity scores when their level began. But activity scores are continually being updated, so the old ones might be considerably out of touch with present realities. For example, we've just boosted ACT(53), ACT(27), ACT(36), ACT(70), ..., in the process of learning (116) — see (115). Thus it's quite possible that several of the first 17 decisions no longer seem wise, because those literals haven't participated in any recent conflicts.

Let x_k be a variable with maximum ACT(k), among all of the variables not in the current trail. It's easy to find such a k (see exercise 290). Now consider, as a thought experiment, what would happen if we were to jump back all the way to level 0 at this point and start over. Recall that our phase-saving strategy dictates that we would set OVAL(j) $\leftarrow$ VAL(j) just before setting VAL(j) $\leftarrow -1$, as the variables become unassigned.

If we now restart at step C6 with $d \leftarrow 1$, all variables whose activity exceeds ACT(k) will receive their former values (although not necessarily in the same order), because the corresponding literals will enter the trail either as decisions or as forced propagations. History will more or less repeat itself, because the old assignments did not cause any conflicts, and because phases were saved.

We might as well therefore avoid most of this back-and-forth unsetting and resetting, by reusing the trail and jumping back only partway, to the first level

Flushing literals and restarting
van der Tak
Ramos
Heule
activity scores
ACT(k)
phase-saving
reusing the trail

literal flushing
activity score
clichés
Biere
agility
damping factor
polarity

where the current activity scores significantly change the picture:

$$\text{Set } d' \leftarrow 0. \text{ While } \texttt{ACT}(|L_{i_{d'+1}}|) \geq \texttt{ACT}(k), \text{ set } d' \leftarrow d'+1. \text{ Then if } d' < d, \text{ jump back to level } d'. \qquad (126)$$

This is the technique called "literal flushing," because it removes the literals on levels $d'+1$ through d and leaves the others assigned. It effectively redirects the search into new territory, without being as drastic as a full restart.

In Table 3, for example, ACT(49) might exceed the activity score of every other unassigned variable; and it might also exceed ACT(46), the activity of the decision literal $\overline{46}$ on level 15. If the previous 14 decision-oriented activities ACT(53), ACT(55), ..., ACT(37) are all $\geq$ ACT(49), we would flush all the literals L_{25}, L_{26}, ... above level $d' = 14$, and commence a new level 15.

Notice that some of the flushed literals other than $\overline{46}$ might actually have the largest activities of all. In such cases they will re-insert themselves, before 49 ever enters the scene. Eventually, though, the literal 49 will inaugurate a new level before a new conflict arises. (See exercise 291.)

Experience shows that flushing can indeed be extremely helpful. On the other hand, it can be harmful if it causes us to abandon a fruitful line of attack. When the solver is perking along and learning useful clauses by the dozen, we don't want to upset the applecart by rocking the boat. Armin Biere has therefore introduced a useful statistic called *agility*, which tends to be correlated with the desirability of flushing at any given moment. His idea [*LNCS* **4996** (2008), 28–33] is beautifully simple: We maintain a 32-bit integer variable called AGILITY, initially zero. Whenever a literal l is placed on the trail in steps C4, C6, or C9, we update the agility by setting

$$\texttt{AGILITY} \leftarrow \texttt{AGILITY} - (\texttt{AGILITY} \gg 13) + \big(((\texttt{OVAL}(|l|) - \texttt{VAL}(|l|)) \,\&\, 1) \ll 19\big). \qquad (127)$$

In other words, the fraction $\texttt{AGILITY}/2^{32}$ is essentially multiplied by $1-\delta$, then increased by δ if the new polarity of l differs from its previous polarity, where $\delta = 2^{-13} \approx .0001$. High agility means that lots of the recent propagations are flipping the values of variables and trying new possibilities; low agility means that the algorithm is basically in a rut, spinning its wheels and getting nowhere.

Table 4

TO FLUSH OR NOT TO FLUSH?

Let $a = \texttt{AGILITY}/2^{32}$ when setting $M_{\rm f} \leftarrow M + \Delta_{\rm f}$, and let $\psi = 1/6$, $\theta = 17/16$.

If $\Delta_{\rm f}$ is	then flush if	If $\Delta_{\rm f}$ is	then flush if	If $\Delta_{\rm f}$ is	then flush if
1	$a \leq \psi \approx .17$	32	$a \leq \theta^5\psi \approx .23$	1024	$a \leq \theta^{10}\psi \approx .31$
2	$a \leq \theta\psi \approx .18$	64	$a \leq \theta^6\psi \approx .24$	2048	$a \leq \theta^{11}\psi \approx .32$
4	$a \leq \theta^2\psi \approx .19$	128	$a \leq \theta^7\psi \approx .25$	4096	$a \leq \theta^{12}\psi \approx .34$
8	$a \leq \theta^3\psi \approx .20$	256	$a \leq \theta^8\psi \approx .27$	8192	$a \leq \theta^{13}\psi \approx .37$
16	$a \leq \theta^4\psi \approx .21$	512	$a \leq \theta^9\psi \approx .29$	16384	$a \leq \theta^{14}\psi \approx .39$

Armed with the notion of agility, we can finally state what Algorithm C does when step C5 finds $M \geq M_{\rm f}$: First $M_{\rm f}$ is reset to $M + \Delta_{\rm f}$, where $\Delta_{\rm f}$ is

a power of two determined by the "reluctant doubling" sequence ⟨1, 1, 2, 1, 1, 2, 4, 1, ...⟩; that sequence is discussed below and in exercise 293. Then the agility is compared to a threshold, depending on Δ_f, according to the schedule in Table 4. (The parameter ψ in that table can be raised or lowered, if you want to increase or decrease the amount of flushing.) If the agility is sufficiently small, x_k is found and (126) is performed. Nothing changes if the agility is large or if $d' = d$; otherwise (126) has flushed some literals, using the operations of step C8.

reluctant doubling
heuristic
randomized methods
stochastic local search
SLS: Stochastic local search
Gu
Papadimitriou
debug
random walks–
Papadimitriou

Monte Carlo methods. Let's turn now to a completely different way to approach satisfiability problems, based on finding solutions by totally heuristic and randomized methods, often called *stochastic local search*. We often use such methods in our daily lives, even though there's no guarantee of success. The simplest satisfiability-oriented technique of this kind was introduced by Jun Gu [see *SIGART Bulletin* **3**, 1 (January 1992), 8–12] and by Christos Papadimitriou [*FOCS* **32** (1991), 163–169] as a byproduct of more general studies:

> "Start with any truth assignment. While there are unsatisfied clauses, pick any one, and flip a random literal in it."

Some programmers are known to debug their code in a haphazard manner, somewhat like this approach; and we know that such "blind" changes are foolish because they usually introduce new bugs. Yet this idea does have merit when it is applied to satisfiability, so we shall formulate it as an algorithm:

Algorithm P (*Satisfiability by random walk*). Given m nonempty clauses $C_1 \wedge \cdots \wedge C_m$ on n Boolean variables $x_1 \ldots x_n$, this algorithm either finds a solution or terminates unsuccessfully after making N trials.

P1. [Initialize.] Assign random Boolean values to $x_1 \ldots x_n$. Set $j \leftarrow 0$, $s \leftarrow 0$, and $t \leftarrow 0$. (We know that s clauses are satisfied after having made t flips.)

P2. [Success?] If $s = m$, terminate successfully with solution $x_1 \ldots x_n$. Otherwise set $j \leftarrow (j \bmod m)+1$. If clause C_j is satisfied by $x_1 \ldots x_n$, set $s \leftarrow s+1$ and repeat this step.

P3. [Done?] If $t = N$, terminate unsuccessfully.

P4. [Flip one bit.] Let clause C_j be $(l_1 \vee \cdots \vee l_k)$. Choose a random index $i \in \{1, \ldots, k\}$, and change variable $|l_i|$ so that literal l_i becomes true. Set $s \leftarrow 1$, $t \leftarrow t + 1$, and return to P2. ▮

Suppose, for example, that we're given the seven clauses R' of (7). Thus $m = 7$, $n = 4$; and there are two solutions, 01∗1. In this case every nonsolution violates a unique clause; for example, 1100 violates the clause $\bar{1}\bar{2}3$, so step P4 is equally likely to change 1100 to 0100, 1000, or 1110, only one of which is closer to a solution. An exact analysis (see exercise 294) shows that Algorithm P will find a solution after making 8.25 flips, on the average. That's no improvement over a brute-force search through all $2^n = 16$ possibilities; but a small example like this doesn't tell us much about what happens when n is large.

Papadimitriou observed that Algorithm P is reasonably effective when it's applied to 2SAT problems, because each flip has roughly a 50-50 chance of making

Schöning
ballot number

progress in that case. Several years later, Uwe Schöning [*Algorithmica* **32** (2002), 615–623] discovered that the algorithm also does surprisingly well on instances of 3SAT, even though the flips when $k > 2$ in step P4 tend to go "the wrong way":

Theorem U. *If the given clauses are satisfiable, and if each clause has at most three literals, Algorithm P will succeed with probability* $\Omega\big((3/4)^n/n\big)$ *after making at most* n *flips.*

Proof. By complementing variables, if necessary, we can assume that $0\ldots0$ is a solution; under this assumption, every clause has at least one negative literal. Let $X_t = x_1 + \cdots + x_n$ be the number of 1s after t flips have been made. Each flip changes X_t by ±1, and we want to show that there's a nontrivial chance that X_t will become 0. After step P1, the random variable X_0 will be equal to q with probability $\binom{n}{q}/2^n$.

A clause that contains three negative literals is good news for Algorithm P, because it is violated only when all three variables are 1; a flip will *always* decrease X_t in such a case. Similarly, a violated clause with two negatives and one positive will invoke a flip that makes progress 2/3 of the time. The worst case occurs only when a problematic clause has only one negative literal. Unfortunately, every clause might belong to this worst case, for all we know.

Instead of studying X_t, which depends on the pattern of clauses, it's much easier to study another random variable Y_t defined as follows: Initially $Y_0 = X_0$; but $Y_{t+1} = Y_t - 1$ only when step P4 flips the negative literal that has the smallest subscript; otherwise $Y_{t+1} = Y_t + 1$. For example, after taking care of a violated clause such as $x_3 \vee \bar{x}_5 \vee \bar{x}_8$, we have $X_{t+1} = X_t + (+1, -1, -1)$ but $Y_{t+1} = Y_t + (+1, -1, +1)$ in the three possible cases. Furthermore, if the clause contains fewer than three literals, we penalize Y_{t+1} even more, by allowing it to be $Y_t - 1$ only with probability 1/3. (After a clause such as $x_4 \vee \bar{x}_6$, for instance, we put $Y_{t+1} = Y_t - 1$ in only 2/3 of the cases when x_6 is flipped; otherwise $Y_{t+1} = Y_t + 1$.)

We clearly have $X_t \le Y_t$ for all t. Therefore $\Pr(X_t = 0) \ge \Pr(Y_t = 0)$, after t flips have been made; and we've defined things so that it's quite easy to calculate $\Pr(Y_t = 0)$, because Y_t doesn't depend on the current clause j:

$$\Pr(Y_{t+1} = Y_t - 1) = 1/3 \quad\text{and}\quad \Pr(Y_{t+1} = Y_t + 1) = 2/3 \quad\text{when}\quad Y_t > 0.$$

Indeed, the theory of random walks developed in Section 7.2.1.6 tells us how to count the number of scenarios that begin with $Y_0 = q$ and end with $Y_t = 0$, after Y_t has increased p times and decreased $p + q$ times while remaining positive for $0 \le t < 2p + q$. It is the "ballot number" of Eq. 7.2.1.6–(23),

$$C_{p,p+q-1} \;=\; \frac{q}{2p+q}\binom{2p+q}{p}. \tag{128}$$

The probability that $Y_0 = q$ and that $Y_t = 0$ for the first time when $t = 2p + q$ is therefore exactly

$$f(p,q) \;=\; \frac{1}{2^n}\binom{n}{q}\frac{q}{2p+q}\binom{2p+q}{p}\Big(\frac{1}{3}\Big)^{p+q}\Big(\frac{2}{3}\Big)^{p}. \tag{129}$$

Every value of p and q gives a lower bound for the probability that Algorithm P succeeds; and exercise 296 shows that we get the result claimed in Theorem U by choosing $p = q \approx n/3$. ▮

Theorem U might seem pointless, because it predicts success only with exponentially small probability when $N = n$. But if at first we don't succeed, we can try and try again, by repeating Algorithm P with different random choices. And if we repeat it $Kn(4/3)^n$ times, for large enough K, we're almost certain to find a solution unless the clauses can't all be satisfied.

In fact, even more is true, because the proof of Theorem U doesn't exploit the full power of Eq. (129). Exercise 297 carries the analysis further, in a particularly instructive way, and proves a much sharper result:

Corollary W. *When Algorithm P is applied $K(4/3)^n$ times with $N = 2n$ to a set of satisfiable ternary clauses, its success probability exceeds $1 - e^{-K/2}$.* ▮

If the clauses $C_1 \wedge \cdots \wedge C_m$ are unsatisfiable, Algorithm P will never demonstrate that fact conclusively. But if we repeat it $100(4/3)^n$ times and get no solution, Corollary W tells us that the chances of satisfiability are incredibly small (less than 10^{-21}). So it's a safe bet that no solution exists in such a case.

Thus Algorithm P has a surprisingly good chance of finding solutions "with its eyes closed," while walking at random in the gigantic space of all 2^n binary vectors; and we can well imagine that even better results are possible if we devise randomized walking methods that proceed with eyes wide open. Therefore many people have experimented with strategies that try to make intelligent choices about which direction to take at each flip-step. One of the simplest and best of these improvements, popularly known as WalkSAT, was devised by B. Selman, H. A. Kautz, and B. Cohen [*Nat. Conf. Artificial Intelligence* **12** (1994), 337–343]:

Algorithm W (*WalkSAT*). Given m nonempty clauses $C_1 \wedge \cdots \wedge C_m$ on n Boolean variables $x_1 \ldots x_n$, and a "greed-avoidance" parameter p, this algorithm either finds a solution or terminates unsuccessfully after making N trials. It uses auxiliary arrays $c_1 \ldots c_n$, $f_0 \ldots f_{m-1}$, $k_1 \ldots k_m$, and $w_1 \ldots w_m$.

W1. [Initialize.] Assign random Boolean values to $x_1 \ldots x_n$. Also set $r \leftarrow t \leftarrow 0$ and $c_1 \ldots c_n \leftarrow 0 \ldots 0$. Then, for $1 \le j \le m$, set k_j to the number of true literals in C_j; and if $k_j = 0$, set $f_r \leftarrow j$, $w_j \leftarrow r$, and $r \leftarrow r + 1$; or if $k_j = 1$ and the only true literal of C_j is x_i or $\bar{x}_i$, set $c_i \leftarrow c_i + 1$. (Now r is the number of unsatisfied clauses, and the f array lists them. The number c_i is the "cost" or "break count" for variable x_i, namely the number of additional clauses that will become false if x_i is flipped.)

W2. [Done?] If $r = 0$, terminate successfully with solution $x_1 \ldots x_n$. Otherwise, if $t = N$, terminate unsuccessfully.

W3. [Choose j.] Set $j \leftarrow f_q$, where q is uniformly random in $\{0, 1, \ldots, r-1\}$. (In other words, choose an unsatisfied clause C_j at random, considering every such clause to be equally likely; exercise 3.4.1–3 discusses the best way to compute q.) Let clause C_j be $(l_1 \vee \cdots \vee l_k)$.

n-cube
WalkSAT–
Selman
Kautz
Cohen
break count

W4. [Choose l.] Let c be the smallest cost among the literals $\{l_1, \ldots, l_k\}$. If $c = 0$, or if $c \geq 1$ and $U \geq p$ where U is uniform in $[0 \mathinner{..} 1)$, choose l randomly from among the literals of cost c. (We call this a "greedy" choice, because flipping l will minimize the number of newly false clauses.) Otherwise choose l randomly in $\{l_1, \ldots, l_k\}$.

W5. [Flip l.] Change the value of variable $|l|$, and update r, $c_1 \ldots c_n$, $f_0 \ldots f_{r-1}$, $k_1 \ldots k_m$, $w_1 \ldots w_m$ to agree with this new value. (Exercise 302 explains how to implement steps W4 and W5 efficiently, with computer-friendly changes to the data structures.) Set $t \leftarrow t + 1$ and return to W2. ▮

Seitz
Alava
Orponen
3SAT
2SAT
reluctant doubling
Luby
Sinclair
Zuckerman

If, for example, we try to satisfy the seven clauses of (7) with Algorithm W, as we did earlier with Algorithm P, the choice $x_1x_2x_3x_4 = 0110$ violates $\bar{2}\bar{3}4$; and $c_1c_2c_3c_4$ turns out to be 0110 in this situation. So step W4 will choose to flip x_4, and we'll have the solution 0111. (See exercise 303.)

Notice that step W3 focuses attention on variables that need to change. Furthermore, a literal that appears in the most unsatisfied clauses is most likely to appear in the chosen clause C_j.

If no cost-free flip is available, step W4 makes nongreedy choices with probability p. This policy keeps the algorithm from getting stuck in an unsatisfiable region from which there's no greedy exit. Extensive experiments by S. Seitz, M. Alava, and P. Orponen [*J. Statistical Mechanics* (June 2005), P06006:1–27] indicate that the best choice of p is .57 when large random 3SAT problems are being tackled. For example, with this setting of p, and with $m = 4.2n$ random 3-literal clauses, Algorithm W works fantastically well: It tends to find solutions after making fewer than 10,000n flips when $n = 10^4$, and fewer than 2500n flips when $10^5 \leq n \leq 10^6$.

What about the parameter N? Should we set it equal to $2n$ (as recommended for 3SAT problems with respect to Algorithm P), or perhaps to n^2 (as recommended for 2SAT in exercise 299), or to 2500n (as just mentioned for 3SAT in Algorithm W), or to something else? When we use an algorithm like WalkSAT, whose behavior can vary wildly depending on random choices and on unknown characteristics of the data, it's often wise to "cut our losses" and to start afresh with a brand new pattern of random numbers.

Exercise 306 proves that such an algorithm always has an optimum cutoff value $N = N^*$, which minimizes the expected time to success when the algorithm is restarted after each failure. Sometimes $N^* = \infty$ is the best choice, meaning that we should always keep plowing ahead; in other cases N^* is quite small.

But N^* exists only in theory, and the theory requires perfect knowledge of the algorithm's behavior. In practice we usually have little or no information about how N should best be specified. Fortunately there's still an effective way to proceed, by using the notion of *reluctant doubling* introduced by M. Luby, A. Sinclair, and D. Zuckerman [*Information Proc. Letters* **47** (1993), 173–180], who defined the interesting sequence

$$S_1, S_2, \ldots = 1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, 1, 1, 2, 1, 1, 2, 4, 1, 1, 2, \ldots. \quad (130)$$

The elements of this sequence are all powers of 2. Furthermore we have $S_{n+1} = 2S_n$ if the number S_n has already occurred an even number of times, otherwise $S_{n+1} = 1$. A convenient way to generate this sequence is to work with two integers (u, v), and to start with $(u_1, v_1) = (1, 1)$; then

$$(u_{n+1}, v_{n+1}) \;=\; \bigl(u_n \,\&\, {-u_n} = v_n?\; (u_n + 1, 1)\colon (u_n, 2v_n)\bigr). \qquad (131)$$

The successive pairs are $(1,1)$, $(2,1)$, $(2,2)$, $(3,1)$, $(4,1)$, $(4,2)$, $(4,4)$, $(5,1)$, $\ldots$, and we have $S_n = v_n$ for all $n \ge 1$.

The reluctant doubling strategy is to run Algorithm W repeatedly with $N = cS_1,\ cS_2,\ cS_3,\ \ldots,$ until success is achieved, where c is some constant. Exercise 308 proves that the expected running time X obtained in this way exceeds the optimum by at most a factor of $O(\log X)$. Other sequences besides $\langle S_n \rangle$ also have this property, and they're sometimes better (see exercise 311). The best policy is probably to use $\langle cS_n \rangle$, where c represents our best guess about the value of N^*; in this way we hedge our bets in case c is too small.

restart schedule
Local Lemma–
probabilistic method
Erdős
clique
random graph
Ramsey's theorem
Lovász
Erdős
Lovász
Spencer

The Local Lemma. The existence of particular combinatorial patterns is often established by using a nonconstructive proof technique called the "probabilistic method," pioneered by Paul Erdős. If we can show that $\Pr(X) > 0$, in some probability space, then X must be true in at least one case. For example [*Bull. Amer. Math. Soc.* **53** (1947), 292–294], Erdős famously observed that there is a graph G on n vertices such that neither G nor $\overline{G}$ contains a k-clique, whenever

$$\binom{n}{k} \;<\; 2^{k(k-1)/2-1}. \qquad (132)$$

For if we consider a random graph G, each of whose $\binom{n}{2}$ edges is present with probability $1/2$, and if U is any particular subset of k vertices in G, the probability that either $G \,|\, U$ or $\overline{G} \,|\, U$ is a complete graph is clearly $2/2^{k(k-1)/2}$. Hence the probability that this doesn't happen for any of the $\binom{n}{k}$ subsets U is at least $1 - \binom{n}{k} 2^{1-k(k-1)/2}$. This probability is positive; so such a graph must exist.

The proof just given does not provide any explicit construction. But it does show that we can find such a graph by making at most $1/\bigl(1 - \binom{n}{k} 2^{1-k(k-1)/2}\bigr)$ random trials, on the average, provided that n and k are small enough that we are able to test all $\binom{n}{k}$ subgraphs in a reasonable amount of time.

Probability calculations of this kind are often complicated by dependencies between the random events being considered. For example, the presence of a clique in one part of a graph affects the likelihood of many other cliques that share some of the same vertices. But the interdependencies are often highly localized, so that "remote" events are essentially independent of each other. László Lovász introduced an important way to deal with such situations early in the 1970s, and his approach has become known as the "Local Lemma" because it has been used to establish many theorems. First published as a lemma on pages 616–617 of a longer paper [Erdős and Lovász, *Infinite and Finite Sets*, *Colloquia Math. Soc. János Bolyai* **10** (1975), 609–627], and subsequently extended to a "lopsided" form [P. Erdős and J. Spencer, *Discrete Applied Math.* **30** (1991), 151–154], it can be stated as follows:

$\mathcal{R}(G)$
lopsidependency graph
dependency graph
Lovász
Spencer
Shearer
Moser
Tardos

Lemma L. *Let $A_1, \ldots, A_m$ be events in some probability space. Let G be a graph on vertices $\{1,\ldots,m\}$, and let $(p_1,\ldots,p_m)$ be numbers such that*

$$\Pr(A_i \mid \overline{A}_{j_1} \cap \cdots \cap \overline{A}_{j_k}) \;\le\; p_i \text{ whenever } k \ge 0 \text{ and } i \not\frown j_1, \ldots, i \not\frown j_k. \quad (133)$$

Then $\Pr(\overline{A}_1 \cap \cdots \cap \overline{A}_m) > 0$ whenever $(p_1,\ldots,p_m)$ lies in a certain set $\mathcal{R}(G)$. ∎

In applications we think of the A_j as "bad" events, which are undesirable conditions that interfere with whatever we're trying to find. The graph G is called a "lopsidependency graph" for our application; this name was coined as an extension of Lovász's original term "dependency graph," for which the strict condition '$= p_i$' was assumed in place of '$\le p_i$' in (133).

The set $\mathcal{R}(G)$ of probability bounds for which we can guarantee that all bad events can simultaneously be avoided, given (133), will be discussed further below. If G is the complete graph K_m, so that (133) simply states that $\Pr(A_i) \le p_i$, $\mathcal{R}(G)$ is clearly $\{(p_1,\ldots,p_m) \mid (p_1,\ldots,p_m) \ge (0,\ldots,0) \text{ and } p_1+\cdots+p_m < 1\}$; this is the smallest possible $\mathcal{R}(G)$. At the other extreme, if G is the empty graph $\overline{K_m}$, we get $\{(p_1,\ldots,p_m) \mid 0 \le p_j < 1 \text{ for } 1 \le j \le m\}$, the largest possible $\mathcal{R}(G)$. Adding an edge to G makes $\mathcal{R}(G)$ smaller. Notice that, if $(p_1,\ldots,p_m)$ is in $\mathcal{R}(G)$ and $0 \le p'_j \le p_j$ for $1 \le j \le m$, then also $(p'_1,\ldots,p'_m) \in \mathcal{R}(G)$.

Lovász discovered an elegant local condition that suffices to make Lemma L widely applicable [see J. Spencer, *Discrete Math.* **20** (1977), 69–76]:

Theorem L. *The probability vector $(p_1,\ldots,p_m)$ is in $\mathcal{R}(G)$ when there are numbers $0 \le \theta_1,\ldots,\theta_m < 1$ such that*

$$p_i = \theta_i \prod_{i \,—\, j \text{ in } G} (1-\theta_j). \quad (134)$$

Proof. Exercise 344(e) proves that $\Pr(\overline{A}_1 \cap \cdots \cap \overline{A}_m) \ge (1-\theta_1)\ldots(1-\theta_m)$. ∎

James B. Shearer [*Combinatorica* **5** (1985), 241–245] went on to determine the exact maximum extent of $\mathcal{R}(G)$ for all graphs G, as we'll see later; and he also established the following important special case:

Theorem J. *Suppose every vertex of G has degree $\le d$, where $d > 1$. Then $(p,\ldots,p) \in \mathcal{R}(G)$ when $p \le (d-1)^{d-1}/d^d$.*

Proof. See the interesting inductive argument in exercise 317. ∎

This condition on p holds whenever $p \le 1/(ed)$ (see exercise 319).

Further study led to a big surprise: The Local Lemma proves only that desirable combinatorial patterns *exist*, although they might be rare. But Robin Moser and Gábor Tardos discovered [*JACM* **57** (2010), 11:1–11:15] that we can efficiently *compute* a pattern that avoids all of the bad A_j, using an almost unbelievably simple algorithm analogous to WalkSAT!

Algorithm M (*Local resampling*). Given m events $\{A_1,\ldots,A_m\}$ that depend on n Boolean variables $\{x_1,\ldots,x_n\}$, this algorithm either finds a vector $x_1\ldots x_n$ for which none of the events is true, or loops forever. We assume that A_j is a function of the variables $\{x_k \mid k \in \Xi_j\}$ for some given subset $\Xi_j \subseteq \{1,\ldots,n\}$.

Whenever the algorithm assigns a value to x_k, it sets $x_k \leftarrow 1$ with probability ξ_k and $x_k \leftarrow 0$ with probability $1-\xi_k$, where ξ_k is another given parameter.

M1. [Initialize.] For $1 \le k \le n$, set $x_k \leftarrow [U < \xi_k]$, where U is uniform in $[0\,..\,1)$.

M2. [Choose j.] Set j to the index of any event such that A_j is true. If no such j exists, terminate successfully, having found a solution $x_1 \ldots x_n$.

M3. [Resample for A_j.] For each $k \in \Xi_j$, set $x_k \leftarrow [U < \xi_k]$, where U is uniform in $[0\,..\,1)$. Return to M2. ▮

(We have stated Algorithm M in terms of binary variables x_k purely for convenience. The same ideas apply when each x_k has a discrete probability distribution on *any* set of values, possibly different for each k.)

To tie this algorithm to the Local Lemma, we assume that event A_i holds with probability $\le p_i$ whenever the variables it depends on have the given distribution. For example, if A_i is the event "$x_3 \ne x_5$" then p_i must be at least $\xi_3(1-\xi_5)+(1-\xi_3)\xi_5$.

We also assume that there's a graph G on vertices $\{1,\ldots,m\}$ such that condition (133) is true, and that $i \,—\, j$ whenever $i \ne j$ and $\Xi_i \cap \Xi_j \ne \emptyset$. Then G is a suitable dependency graph for $\{A_1,\ldots,A_m\}$, because the events A_{j_1}, $\ldots$, A_{j_k} can't possibly influence A_i when $i \not\!—\, j_1$, $\ldots$, $i \not\!—\, j_k$. (Those events share no common variables with A_i.) We can also sometimes get by with fewer edges by making G a *lopsidependency* graph; see exercise 351.

Algorithm M might succeed with *any* given events, purely by chance. But if the conditions of the Local Lemma are satisfied, success can be guaranteed:

Theorem M. *If* (133) *holds with probabilities that satisfy condition* (134) *of Theorem L, step M3 is performed for A_j at most $\theta_j/(1-\theta_j)$ times, on average.*

Proof. Exercise 352 shows that this result is a corollary of the more general analysis that is carried out below. The stated upper bound is good news, because θ_j is usually quite small. ▮

Traces and pieces. The best way to understand why Algorithm M is so efficient is to view it algebraically in terms of "traces." The theory of traces is a beautiful area of mathematics in which amazingly simple proofs of profound results have been discovered. Its basic ideas were first formulated by P. Cartier and D. Foata [*Lecture Notes in Math.* **85** (1969)], then independently developed from another point of view by R. M. Keller [*JACM* **20** (1973), 514–537, 696–710] and A. Mazurkiewicz ["Concurrent program schemes and their interpretations," DAIMI Report PB 78 (Aarhus University, July 1977)]. Significant advances were made by G. X. Viennot [*Lecture Notes in Math.* **1234** (1985), 321–350], who presented many wide-ranging applications and explained how the theory could readily be visualized in terms of what he called "heaps of pieces."

Trace theory is the study of algebraic products whose variables are not necessarily commutative. Thus it forms a bridge between the study of strings (in which, for example, *acbbaca* is quite distinct from *baccaab*) and the study of ordinary commutative algebra (in which both of those examples are equal to

reliability polynomial
lopsidependency
traces–
Cartier
Foata
Keller
Mazurkiewicz
Viennot
heaps of pieces
commutativity, partial
strings

$aaabbcc = a^3b^2c^2$). Each adjacent pair of letters $\{a, b\}$ either *commutes*, meaning that $ab = ba$, or *clashes*, meaning that ab is different from ba. If, for instance, we specify that a commutes with c but that b clashes with both a and c, then *acbbaca* is equal to *cabbaac*, and it has six variants altogether; similarly, there are ten equally good ways to write *baccaab*.

Formally speaking, a *trace* is an equivalence class of strings that can be converted to each other by repeatedly interchanging pairs of adjacent letters that don't clash. But we don't need to fuss about the fact that equivalence classes are present; we can simply represent a trace by any one of its equivalent strings, just as we don't distinguish between equivalent fractions such as 1/2 and 3/6.

Every graph whose vertices represent distinct letters defines a family of traces on those letters, when we stipulate that two letters clash if and only if they are adjacent in the graph. For example, the path graph a — b — c corresponds to the rules stated above. The distinct traces for this graph are

$$\epsilon, a, b, c, aa, ab, ac, ba, bb, bc, cb, cc, aaa, aab, \ldots, ccb, ccc, aaaa, \ldots \qquad (135)$$

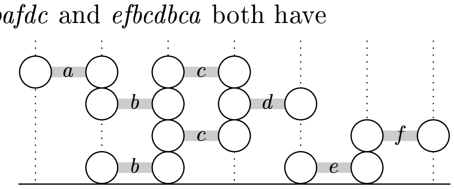
if we list them first by size and then in lexicographic order. (Notice that ca is absent, because ac has already appeared.) The complete graph K_n defines traces that are the same as strings, when *nothing* commutes; the empty graph $\overline{K_n}$ defines traces that are the same as monomials, when *everything* commutes. If we use the path a — b — c — d — e — f to define clashes, the traces *bcebafdc* and *efbcdbca* turn out to be the same.

Viennot observed that partial commutativity is actually a familiar concept, if we regard the letters as "pieces" that occupy "territory." Pieces clash if and only if their territories overlap; pieces commute if and only if their territories are disjoint. A trace corresponds to stacking the pieces on top of one another, from left to right, letting each new piece "fall" until it either rests on the ground or on another piece. In the latter case, it must rest on the most recent piece with which it clashes. He called this configuration an *empilement* — a nice French word.

More precisely, each piece a is assigned a nonempty subset $T(a)$ of some universe, and we say that a clashes with b if and only if $T(a) \cap T(b) \neq \emptyset$. For example, the constraints of the graph a — b — c — d — e — f arise when we let

$$T(a) = \{1, 2\}, \quad T(b) = \{2, 3\}, \quad T(c) = \{3, 4\}, \quad \ldots, \quad T(f) = \{6, 7\};$$

then the traces *bcebafdc* and *efbcdbca* both have



(136)

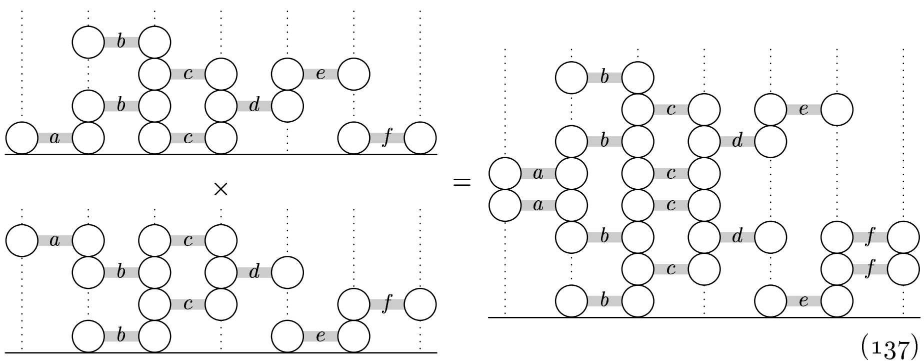
as their empilement. (Readers who have played the game of Tetris® will immediately understand how such diagrams are formed, although the pieces in trace theory differ from those of Tetris because they occupy only a single horizontal level. Furthermore, each type of piece always falls in exactly the same place; and a piece's territory $T(a)$ might have "holes" — it needn't be connected.)

clashing pairs of letters
equivalence class
path graph
lexicographic order
Viennot
territory
pieces–
territories
stacking the pieces
empilement
intersection graph
diagram of a trace
Tetris

Two traces are the same if and only if they have the same empilement. In fact, the diagram implicitly defines a partial ordering on the pieces that appear; and the number of different strings that represent any given trace is the number of ways to sort that ordering topologically (see exercise 324).

Every trace α has a *length*, denoted by $|\alpha|$, which is the number of letters in any of its equivalent strings. It also has a *height*, written $h(\alpha)$, which is the number of levels in its empilement. For example, $|bcebafdc| = 8$ and $h(bcebafdc) = 4$.

Arithmetic on traces. To multiply traces, we simply concatenate them. If, for example, $\alpha = bcebafdc$ is the trace corresponding to (136), then $\alpha\alpha^R = bcebafdccdfabecb$ has the following empilement:



(137)

The algorithm in exercise 327 formulates this procedure precisely. A moment's thought shows that $|\alpha\beta| = |\alpha| + |\beta|$, $h(\alpha\beta) \le h(\alpha) + h(\beta)$, and $h(\alpha\alpha^R) = 2h(\alpha)$.

Traces can also be *divided*, in the sense that $\alpha = (\alpha\beta)/\beta$ can be determined uniquely when $\alpha\beta$ and β are given. All we have to do is remove the pieces of β from the pieces of $\alpha\beta$, one by one, working our way down from the top of the empilements. Similarly, the value of $\beta = \alpha \setminus (\alpha\beta)$ can be computed from the traces α and $\alpha\beta$. (See exercises 328 and 329.)

Notice that we could rotate diagrams like (136) and (137) by 90 degrees, thereby letting the pieces "fall" to the *left* instead of downwards. (We've used a left-to-right approach for similar purposes in Section 5.3.4, Fig. 50.) Or we could let them fall upwards, or to the right. Different orientations are sometimes more natural, depending on what we're trying to do.

We can also add and subtract traces, thereby obtaining polynomials in variables that are only partially commutative. Such polynomials can be multiplied in the normal way; for example, $(\alpha + \beta)(\gamma - \delta) = \alpha\gamma - \alpha\delta + \beta\gamma - \beta\delta$. Indeed, we can even work with *infinite* sums, at least formally: The generating function for all traces that belong to the graph $a — b — c$ is

$$1+a+b+c+aa+ab+ac+ba+bb+bc+cb+cc+aaa+\cdots+ccc+aaaa+\cdots. \quad (138)$$

(Compare with (135); we now use 1, not ϵ, to stand for the empty string.)

The infinite sum (138) can actually be expressed in closed form: It equals

$$\frac{1}{1-a-b-c+ac} = 1 + (a+b+c-ac) + (a+b+c-ac)^2 + \cdots, \quad (139)$$

topological sorting
length of a trace
height of a trace
multiplication of traces
division of traces
right division of traces
polynomials
generating function
empty string

an identity that is correct not only when the variables are commutative, but also in the algebra of traces, when variables commute only when they don't clash.

In their original monograph of 1969, Cartier and Foata showed that the sum of all traces with respect to *any* graph can be expressed in a remarkably simple way that generalizes (139). Let's define the *Möbius function* of a trace α with respect to a graph G by the rule

$$\mu_G(\alpha) \;=\; \begin{cases} 0, & \text{if } h_G(\alpha) > 1; \\ (-1)^{|\alpha|}, & \text{otherwise.} \end{cases} \qquad (140)$$

(The classical Möbius function $\mu(n)$ for integers, defined in exercise 4.5.2–10, is analogous.) Then the *Möbius series* for G is defined to be

$$M_G \;=\; \sum_{\alpha} \mu_G(\alpha)\alpha, \qquad (141)$$

where the sum is over all traces. This sum is a polynomial, when G is finite, because it contains exactly one nonzero term for every independent set of vertices in G; therefore we might call it the *Möbius polynomial*. For example, when G is the path $a \mathbin{—} b \mathbin{—} c$, we have $M_G = 1 - a - b - c + ac$, the denominator in (139). Cartier and Foata's generalization of (139) has a remarkably simple proof:

Theorem F. *The generating function T_G for the sum of all traces, with respect to any graph G, is $1/M_G$.*

Proof. We want to show that $M_G T_G = 1$, in the (partially commutative) algebra of traces. This infinite product is $\sum_{\alpha,\beta} \mu_G(\alpha) = \sum_\gamma \sum_{\alpha,\beta} \mu_G(\alpha)[\gamma = \alpha\beta]$. Hence we want to show that the sum of $\mu_G(\alpha)$, over all ways to factorize $\gamma = \alpha\beta$ as the product of two traces α and β, is zero whenever γ is nonempty.

But that's easy. We can assume that the letters are ordered in some arbitrary fashion. Let a be the smallest letter in the bottom level of γ's empilement. We can restrict attention to cases where α consists of independent (commuting) letters (pieces), because $\mu_G(\alpha) = 0$ otherwise. Now if $\alpha = a\alpha'$ for some trace α', let $\beta' = a\beta$; otherwise we must have $\beta = a\beta'$ for some trace β', and we let $\alpha' = a\alpha$. In both cases $\alpha\beta = \alpha'\beta'$, $(\alpha')' = \alpha$, $(\beta')' = \beta$, and $\mu_G(\alpha) + \mu_G(\alpha') = 0$. So we've grouped all possible factorizations of γ into pairs that cancel out in the sum. ▮

The Möbius series for any graph can be computed recursively via the formula

$$M_G \;=\; M_{G\setminus a} - aM_{G\setminus a^*}, \qquad a^* = \{a\} \cup \{b \mid a \mathbin{—} b\}, \qquad (142)$$

where a is any letter (vertex) of G, because we have $a \notin I$ or $a \in I$ whenever I is independent. For example, if G is the path $a \mathbin{—} b \mathbin{—} c \mathbin{—} d \mathbin{—} e \mathbin{—} f$, then $G \setminus a^* = G \mid \{c, d, e, f\}$ is the path $c \mathbin{—} d \mathbin{—} e \mathbin{—} f$; repeated use of (142) yields

$$\begin{aligned} M_G \;=\; 1 - a - b - c - d - e - f + ac + ad + ae + af \\ + bd + be + bf + ce + cf + df - ace - acf - adf - bdf \end{aligned} \qquad (143)$$

in this case. Since M_G is a polynomial, we can indicate its dependence on the variables by writing $M_G(a, b, c, d, e, f)$. Notice that M_G is always multilinear (this is, linear in each variable); and $M_{G\setminus a}(b, c, d, e, f) = M_G(0, b, c, d, e, f)$.

Cartier
Foata
Möbius function of a trace
Möbius function
Möbius series
Möbius polynomial
multilinear

In applications we often want to replace each letter in the polynomial by a single variable, such as z, and write $M_G(z)$. The polynomial in (143) then becomes $M_G(z) = 1-6z+10z^2-4z^3$; and we can conclude from Theorem F that the number of traces of length n with respect to G is $[z^n]\,1/(1-6z+10z^2-4z^3) = \frac{1}{4}(2+\sqrt{2})^{n+2} + \frac{1}{4}(2-\sqrt{2})^{n+2} - 2^{n+1}$.

Although (142) is a simple recurrence for M_G, we can't conclude that M_G is easy to compute when G is a large and complicated graph. Indeed, the degree of M_G is the size of a maximum independent set in G; and it's NP-hard to determine that number! On the other hand, there are many classes of graphs, such as interval graphs and forests, for which M_G can be computed in linear time.

If α is any trace, the letters that can occur first in a string that represents it are called the *sources* of α; these are the pieces on the bottom level of α's empilement, also called its minimal pieces. Dually, the letters that can occur last are the *sinks* of α, its maximal pieces. A trace that has only one source is called a *cone*; in this case all pieces are ultimately supported by a single piece at the bottom. A trace that has only one sink is, similarly, called a *pyramid*. Viennot proved a nice generalization of Theorem F in his lecture notes:

$$M_{G\setminus A}/M_G \text{ is the sum of all traces whose sources are contained in } A. \qquad (144)$$

(See exercise 338; Theorem F is the special case where A is the set of all vertices.) In particular, the cones for which a is the only source are generated by

$$M_{G\setminus a}/M_G - 1 \;=\; a M_{G\setminus a^*}/M_G. \qquad (145)$$

***Traces and the Local Lemma.** Now we're ready to see why the theory of traces is intimately connected with the Local Lemma. If G is any graph on the vertices $\{1,\ldots,m\}$, we say that $\mathcal{R}(G)$ is the set of all nonnegative vectors $(p_1,\ldots,p_m)$ such that $M_G(p'_1,\ldots,p'_m) > 0$ whenever $0 \le p'_j \le p_j$ for $1 \le j \le m$. This definition of $\mathcal{R}(G)$ is consistent with the implicit definition already given in Lemma L, because of the following characterization found by J. B. Shearer:

Theorem S. *Under condition* (133) *of Lemma L,* $(p_1,\ldots,p_m) \in \mathcal{R}(G)$ *implies*

$$\Pr(\overline{A}_1 \cap \cdots \cap \overline{A}_m) \ge M_G(p_1,\ldots,p_m) > 0. \qquad (146)$$

Conversely, if $(p_1,\ldots,p_m) \notin \mathcal{R}(G)$*, there are events* $B_1, \ldots, B_m$ *such that*

$$\Pr(B_i \mid \overline{B}_{j_1} \cap \cdots \cap \overline{B}_{j_k}) \;=\; p_i \text{ whenever } k \ge 0 \text{ and } i \not\!\!-\!\!- j_1, \ldots, i \not\!\!-\!\!- j_k, \qquad (147)$$

and $\Pr(\overline{B}_1 \cap \cdots \cap \overline{B}_m) = 0$.

Proof. When $(p_1,\ldots,p_m) \in \mathcal{R}(G)$, exercise 344 proves that there's a unique distribution for events $B_1, \ldots, B_m$ such that they satisfy (147) and also

$$\Pr\Bigl(\bigcap_{j\in J} \overline{A}_j\Bigr) \;\ge\; \Pr\Bigl(\bigcap_{j\in J} \overline{B}_j\Bigr) \;=\; M_G\bigl(p_1[1\in J],\ldots,p_m[m\in J]\bigr) \qquad (148)$$

for every subset $J \subseteq \{1,\ldots,m\}$. In this "extreme" worst-possible distribution, $\Pr(B_i \cap B_j) = 0$ whenever $i \mathrel{-\!\!-} j$ in G. Exercise 345 proves the converse. ▮

maximum independent set
NP-hard
interval graphs
forests
sources
sinks
cone
pyramid
Viennot
$\mathcal{R}(G)$
Shearer

Pringsheim
slack
consecutive 1s

Given a probability vector $(p_1, \ldots, p_m)$, let

$$M_G^*(z) = M_G(p_1 z, \ldots, p_m z). \tag{149}$$

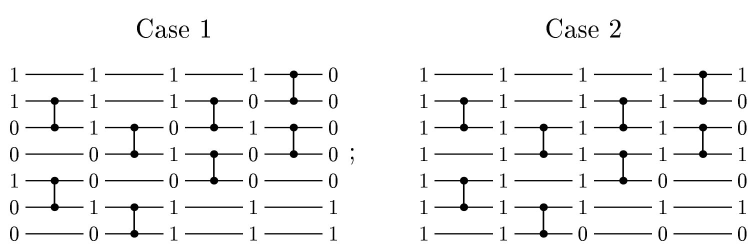
Theorem F tells us that the coefficient of z^n in the power series $1/M_G^*(z)$ is the sum of all traces of length n for G. Since this coefficient is nonnegative, we know by Pringsheim's theorem (see exercise 348) that the power series converges for all $z < 1 + \delta$, where $1 + \delta$ is the smallest real root of the polynomial equation $M_G^*(z) = 0$; this number δ is called the *slack* of $(p_1, \ldots, p_m)$ with respect to G.

It's easy to see that $(p_1, \ldots, p_m) \in \mathcal{R}(G)$ if and only if the slack is positive. For if $\delta \le 0$, the probabilities $(p'_1, \ldots, p'_m)$ with $p'_j = (1 + \delta)p_j$ make $M_G = 0$. But if $\delta > 0$, the power series converges when $z = 1$. And (since it represents the sum of all traces) it also converges to the positive number $1/M_G$ if any p_j is decreased; hence $(p_1, \ldots, p_m)$ lies in $\mathcal{R}(G)$ by definition. Indeed, this argument shows that, when $(p_1, \ldots, p_m) \in \mathcal{R}(G)$, we can actually *increase* the probabilities to $((1 + \epsilon)p_1, \ldots, (1 + \epsilon)p_m)$, and they will still lie in $\mathcal{R}(G)$ whenever $\epsilon < \delta$.

Let's return now to Algorithm M. Suppose the successive bad events A_j that step M3 tries to quench are $X_1, X_2, \ldots, X_N$, where N is the total number of times step M3 is performed (possibly $N = \infty$). To prove that Algorithm M is efficient, we shall show that this random variable N has a small expected value, in the probability space of the independent uniform deviates U that appear in steps M1 and M3. The main idea is that $X_1 X_2 \ldots X_N$ is essentially a trace for the underlying graph; hence we can consider it as an empilement of pieces.

Some simple and concrete examples will help to develop our intuition; we shall consider two case studies. In both cases there are $m = 6$ events A, B, C, D, E, F, and there are $n = 7$ variables $x_1 \ldots x_7$. Each variable is a random bit; thus $\xi_1 = \cdots = \xi_7 = 1/2$ in the algorithm. Event A depends on $x_1 x_2$, while B depends on $x_2 x_3$, …, and F depends on $x_6 x_7$. Furthermore, each event occurs with probability $1/4$. In Case 1, each event is true when its substring is '10'; thus all events are false if and only if $x_1 \ldots x_7$ is sorted — that is, $x_1 \le x_2 \le \cdots \le x_7$. In Case 2, each event is true when its substring is '11'; thus all events are false if and only if $x_1 \ldots x_7$ has no two consecutive 1s.

What happens when we apply Algorithm M to those two cases? One possible scenario is that step M3 is applied $N = 8$ times, with $X_1 X_2 \ldots X_8 = BCEBAFDC$. The actual changes to the bits $x_1 \ldots x_7$ might then be



(150)

(Read $x_1 \ldots x_7$ from top to bottom in these diagrams, and scan from left to right. Each module '$\overset{\bullet}{\underset{\bullet}{|}}$' means "replace the two bad bits at the left by two random bits

at the right.” In examples such as this, any valid solution $x_1 \ldots x_7$ can be placed at the far right; all values to the left of the modules are then forced.)

Notice that these diagrams are like the empilement (136), except that they've been rotated 90°. We know from (136) that the same diagram applies to the scenario *EFBCDBCA* as well as to *BCEBAFDC*, because they're the same, as traces. Well . . . , not quite! In truth, *EFBCDBCA* doesn't give exactly the same result as *BCEBAFDC* in Algorithm M, if we execute that algorithm as presently written. But the results *would* be identical if we used *separate* streams of independent random numbers U_k for each variable x_k. Thus we can legitimately *equate* equivalent traces, in the probability space of our random events.

The algorithm runs much faster in practice when it's applied to Case 1 than when it's applied to Case 2. How can that be? Both of the diagrams in (150) occur with the same probability, namely $(1/2)^7(1/4)^8$, as far as the random numbers are concerned. And every diagram for Case 1 has a corresponding diagram for Case 2; so we can't distinguish the cases by the number of different diagrams. The real difference comes from the fact that, in Case 1, we never have two events to choose from in step M2, unless they are disjoint and can be handled in either order. In Case 2, by contrast, we are deluged at almost every step with events that need to be snuffed out. Therefore the scenario at the right of (150) is actually quite unlikely; why should the algorithm pick B as the first event to correct, and then C, rather than A? Whatever method is used in step M2, we'll find that the diagrams for Case 2 will occur less frequently than dictated by the strict probabilities, because of the decreasing likelihood that any particular event will be worked on next, in the presence of competing choices. (See also exercise 353.)

Worst-case upper bounds on the running time of Algorithm M therefore come from situations like Case 1. In general, the empilement *BCEBAFDC* in (150) will occur in a run of Algorithm M with probability at most *bcebafdc*, if we write 'a' for the probabilistic upper bound for event A that is denoted by 'p_i' in (133) when A is A_i, and if 'b', . . . , 'f' are similar for B, . . . , F. The reason is that *bcebafdc* is clearly the probability that those events are produced by the independent random variables x_k set by the algorithm, if the layers of the corresponding empilement are defined by dependencies between the variable sets Ξ_j. And even if events in the same layer are dependent (by shared variables) yet not lopsidependent (in the sense of exercise 351), such events are positively correlated; so the FKG inequality of exercise MPR–61, which holds for the Bernoulli-distributed variables of Algorithm M, shows that *bcebafdc* is an upper bound. Furthermore the probability that step M2 actually chooses B, C, E, B, A, F, D, and C to work on is at most 1.

Therefore, when $(p_1, \ldots, p_m) \in \mathcal{R}(G)$, Algorithm M's running time is maximized when it is applied to events B_1, . . . , B_m that have the extreme distribution (148) of exercise 344. And we can actually write down the *generating function* for the running time with respect to those extreme events: We have

$$\sum_{N\ge 0} \Pr(\text{Algorithm M on } B_1, \ldots, B_m \text{ does } N \text{ resamplings})\, z^N = \frac{M_G^*(1)}{M_G^*(z)}, \quad (151)$$

handwaving
FKG inequality
Bernoulli-distributed
extreme distribution
generating function

Kolipaka
Szegedy
message passing–
statistical mechanics
bipartite structure
survey propagation–
Braunstein
Mézard
Zecchina

where $M_G^*(z)$ is defined in (149), because the coefficient of z^N in $1/M_G^*(z)$ is the sum of the probabilities of all the traces of length N. Theorem F describes the meaning of $1/M_G^*(1)$ as a "formal" power series in the variables p_i; we proved it without considering whether or not the infinite sum converges when those variables receive numerical values. But when $(p_1, \ldots, p_m) \in \mathcal{R}(G)$, this series is indeed convergent (it even has a positive "slack").

This reasoning leads to the following theorem of K. Kolipaka and M. Szegedy [*STOC* **43** (2011), 235–243]:

Theorem K. *If $(p_1, \ldots, p_m) \in \mathcal{R}(G)$, Algorithm M resamples Ξ_j at most*

$$E_j \;=\; p_j\, M_{G \setminus A_j^*}(p_1, \ldots, p_m)/M_G(p_1, \ldots, p_m) \tag{152}$$

times, on the average. In particular, the expected number of iterations of step M3 is at most $E_1 + \cdots + E_m \le m/\delta$, where δ is the slack of $(p_1, \ldots, p_m)$.

Proof. The extreme distribution B_1, ..., B_m maximizes the number of times Ξ_j is resampled, and the generating function for that number in the extreme case is

$$\frac{M_G(p_1, \ldots, p_{j-1}, p_j, p_{j+1}, \ldots, p_m)}{M_G(p_1, \ldots, p_{j-1}, p_j z, p_{j+1}, \ldots, p_m)}. \tag{153}$$

Differentiating with respect to z, then setting $z \leftarrow 1$, gives (152), because the derivative of the denominator is $-p_j M_{G \setminus A_j^*}(p_1, \ldots, p_m)$ by (141).

The stated upper bound on $E_1 + \cdots + E_m$ is proved in exercise 355. ▮

***Message passing.** Physicists who study statistical mechanics have developed a significantly different way to apply randomization to satisfiability problems, based on their experience with the behavior of large systems of interacting particles. From their perspective, a set of Boolean variables whose values are 0 or 1 is best viewed as an ensemble of particles that have positive or negative "spin"; these particles affect each other and change their spins according to local attractions and repulsions, analogous to laws of magnetism. A satisfiability problem can be formulated as a joint probability distribution on spins for which the states of minimum "energy" are achieved precisely when the spins satisfy as many clauses as possible.

In essence, their approach amounts to considering a bipartite structure in which each variable is connected to one or more clauses, and each clause is connected to one or more variables. We can regard both variables and clauses as active agents, who continually tweet to their neighbors in this social network. A variable might inform its clauses that "I think I should probably be true"; but several of those clauses might reply, "I really wish you were false." By carefully balancing these messages against each other, such local interactions can propagate and build up more and more knowledge of distant connections, often converging to a state where the whole network is reasonably happy.

A particular message-passing strategy called *survey propagation* [A. Braunstein, M. Mézard, and R. Zecchina, *Random Structures & Algorithms* **27** (2005),

201–226] has proved to be astonishingly good at solving random satisfiability problems in the "hard" region just before the threshold of unsatisfiability.

random satisfiability problems
threshold of unsatisfiability
Chavas
Furtlehner
Mézard
Zecchina
Braunstein
focus
flexibility coefficients
field

Let C be a clause and let l be one of its literals. A "survey message" $\eta_{C\to l}$ is a fraction between 0 and 1 that represents how urgently C wants l to be true. If $\eta_{C\to l}=1$, the truth of l is desperately needed, lest C be false; but if $\eta_{C\to l}=0$, clause C isn't the least bit worried about the value of variable $|l|$. Initially we set each $\eta_{C\to l}$ to a completely random fraction.

We shall consider an extension of the original survey propagation method [see J. Chavas, C. Furtlehner, M. Mézard, and R. Zecchina, *J. Statistical Mechanics* (November 2005), P11016:1–25; A. Braunstein and R. Zecchina, *Physical Review Letters* **96** (27 January 2006), 030201:1–4], which introduces additional "reinforcement messages" η_l for each literal l. These new messages, which are initially all zero, represent an external force that acts on l. They help to focus the network activity by reinforcing decisions that have turned out to be fruitful.

Suppose v is a variable that appears in just three clauses: positively in A and B, negatively in C. This variable will respond to its incoming messages $\eta_{A\to v}$, $\eta_{B\to v}$, $\eta_{C\to \bar v}$, η_v, and $\eta_{\bar v}$ by computing two "flexibility coefficients," π_v and $\pi_{\bar v}$, using the following formulas:

$$\pi_v=(1-\eta_v)(1-\eta_{A\to v})(1-\eta_{B\to v}),\qquad \pi_{\bar v}=(1-\eta_{\bar v})(1-\eta_{C\to\bar v}).$$

If, for instance, $\eta_v=\eta_{\bar v}=0$ while $\eta_{A\to v}=\eta_{B\to v}=\eta_{C\to\bar v}=2/3$, then $\pi_v=1/9$, $\pi_{\bar v}=1/3$. The π's are essentially dual to the η's, because high urgency corresponds to low flexibility and vice versa. The general formula for each literal l is

$$\pi_l \;=\; (1-\eta_l)\prod_{l\in C}(1-\eta_{C\to l}). \tag{154}$$

Survey propagation uses these coefficients to estimate variable v's tendency to be either 1 (true), 0 (false), or $*$ (wild), by computing three numbers

$$p=\frac{(1-\pi_v)\pi_{\bar v}}{\pi_v+\pi_{\bar v}-\pi_v\pi_{\bar v}},\quad q=\frac{(1-\pi_{\bar v})\pi_v}{\pi_v+\pi_{\bar v}-\pi_v\pi_{\bar v}},\quad r=\frac{\pi_v\pi_{\bar v}}{\pi_v+\pi_{\bar v}-\pi_v\pi_{\bar v}};\tag{155}$$

then $p+q+r=1$, and (p,q,r) is called the "field" of v, representing respectively (truth, falsity, wildness). The field turns out to be $(8/11, 2/11, 1/11)$ in our example above, indicating that v should probably be assigned the value 1. But if $\eta_{A\to v}$ and $\eta_{B\to v}$ had been only $1/3$ instead of $2/3$, the field would have been $(5/17, 8/17, 4/17)$, and we would probably want $v=0$ in order to satisfy clause C. Figure 51 shows lines of constant $p-q$ as a function of π_v and $\pi_{\bar v}$; the most decisive cases ($|p-q|\approx 1$) occur at the lower right and upper left.

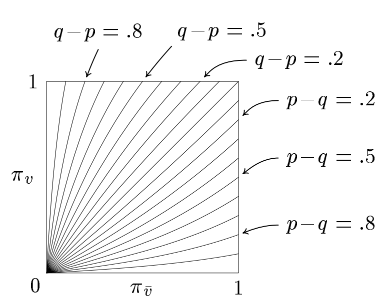


Fig. 51. Lines of constant bias in a variable's "field."

bias message
monus

If $\pi_v = \pi_{\bar v} = 0$, there's no flexibility at all: Variable v is being asked to be both true and false. The field is undefined in such cases, and the survey propagation method hopes that this doesn't happen.

After each literal l has computed its flexibility, the clauses that involve l or $\bar l$ can use π_l and $\pi_{\bar l}$ to refine their survey messages. Suppose, for example, that C is the clause $u \vee \bar v \vee w$. It will replace the former messages $\eta_{C\to u}$, $\eta_{C\to\bar v}$, $\eta_{C\to w}$ by

$$\eta'_{C\to u} = \gamma_{\bar v\to C}\gamma_{w\to C}\,, \qquad \eta'_{C\to\bar v} = \gamma_{u\to C}\gamma_{w\to C}\,, \qquad \eta'_{C\to w} = \gamma_{u\to C}\gamma_{\bar v\to C}\,,$$

where each $\gamma_{l\to C}$ is a "bias message" received from literal l,

$$\gamma_{l\to C} \;=\; \frac{(1-\pi_{\bar l})\pi_l/(1-\eta_{C\to l})}{\pi_{\bar l} + (1-\pi_{\bar l})\pi_l/(1-\eta_{C\to l})}, \tag{156}$$

reflecting l's propensity to be false in clauses other than C. In general we have

$$\eta'_{C\to l} \;=\; \Bigl(\prod_{l'\in C}\gamma_{l'\to C}\Bigr)\Big/\gamma_{l\to C}. \tag{157}$$

(Appropriate conventions must be used to avoid division by zero in formulas (156) and (157); see exercise 359.)

New reinforcement messages η'_l can also be computed periodically, using the formula

$$\eta'_l \;=\; \frac{\kappa(\pi_{\bar l} \dot- \pi_l)}{\pi_l + \pi_{\bar l} - \pi_l\pi_{\bar l}} \tag{158}$$

for each literal l; here $x \dot- y$ denotes $\max(x-y, 0)$, and κ is a reinforcement parameter specified by the algorithm. Notice that $\eta'_l > 0$ only if $\eta'_{\bar l} = 0$.

For example, here are messages that might be passed when we want to satisfy the seven clauses of (7):

l_1	l_2	l_3	$\eta_{C\to l_1}$	$\eta_{C\to l_2}$	$\eta_{C\to l_3}$	$\gamma_{l_1\to C}$	$\gamma_{l_2\to C}$	$\gamma_{l_3\to C}$
1	2	$\bar 3$	0	0	0	3/5	0	0
$\bar 1$	$\bar 2$	3	1/5	0	0	0	3/5	1/3
2	3	$\bar 4$	1/5	0	0	0	1/3	3/5
$\bar 2$	$\bar 3$	4	0	0	0	3/5	0	0
1	3	4	0	0	1/5	3/5	1/3	0
$\bar 1$	$\bar 3$	$\bar 4$	0	0	0	0	0	3/5
$\bar 1$	2	4	0	0	0	0	0	0

l	π_l	η_l
1	1	0
$\bar 1$	2/5	1/2
2	2/5	1/2
$\bar 2$	1	0
3	1	0
$\bar 3$	2/3	1/3
4	2/5	1/2
$\bar 4$	1	0

(159)

(Recall that the only solutions to these clauses are $\bar 1 2 3 4$ and $\bar 1 2 \bar 3 4$.) In this case the reader may verify that the messages of (159) constitute a "fixed point": The η messages determine the π's; conversely, we also have $\eta'_{C\to l} = \eta_{C\to l}$ for all clauses C and all literals l, if the reinforcement messages η_l remain constant.

Exercise 361 proves that every solution to a satisfiable set of clauses yields a fixed point of the simultaneous equations (154), (156), (157), with the property that $\eta_l = [l \text{ is true in the solution}]$.

Experiments with this message-passing strategy have shown, however, that the best results are obtained by using it only for preliminary screening, with the goal of discovering variables whose settings are most critical; we needn't continue to transmit messages until every clause is fully satisfied. Once we've assigned suitable values to the most delicate variables, we're usually left with a residual problem that can readily be solved by other algorithms such as WalkSAT.

The survey, reinforcement, and bias messages can be exchanged using a wide variety of different protocols. The following procedure incorporates two ideas from an implementation prepared by C. Baldassi in 2012: (1) The reinforcement strength κ begins at zero, but approaches 1 exponentially. (2) Variables are rated 1, 0, or $*$ after each reinforcement, according as $\max(p,q,r)$ in their current field is p, q, or r. If every clause then has at least one literal that is true or $*$, message passing will cease even though some surveys might still be fluctuating.

Algorithm S (*Survey propagation*). Given m nonempty clauses on n variables, this algorithm tries to assign values to most of the variables in such a way that the still-unsatisfied clauses will be relatively easy to satisfy. It maintains arrays π_l and η_l of floating-point numbers for each literal l, as well as $\eta_{C\to l}$ for each clause C and each $l \in C$. It has a variety of parameters: ρ (the damping factor for reinforcement), N_0 and N (the minimum and maximum iteration limits), ϵ (the tolerance for convergence), and ψ (the confidence level).

S1. [Initialize.] Set $\eta_l \leftarrow \pi_l \leftarrow 0$ for all literals l, and $\eta_{C\to l} \leftarrow U$ for all clauses C and $l \in C$, where U is uniformly random in $[0\,.\,.\,1)$. Also set $i \leftarrow 0$, $\phi \leftarrow 1$.

S2. [Done?] Terminate unsuccessfully if $i \geq N$. If i is even or $i < N_0$, go to S5.

S3. [Reinforce.] Set $\phi \leftarrow \rho\phi$ and $\kappa \leftarrow 1-\phi$. Replace η_l by η'_l for all literals l, using (158); but terminate unsuccessfully if $\pi_l = \pi_{\bar l} = 0$.

S4. [Test pseudo-satisfiability.] Go to S5 if there is at least one clause whose literals l all appear to be false, in the sense that $\pi_{\bar l} < \pi_l$ and $\pi_{\bar l} < \frac{1}{2}$ (see exercise 358). Otherwise go happily to S8.

S5. [Compute the π's.] Compute each π_l, using (154); see also exercise 359.

S6. [Update the surveys.] Set $\delta \leftarrow 0$. For all clauses C and literals $l \in C$, compute $\eta'_{C\to l}$ using (157), and set $\delta \leftarrow \max\bigl(\delta, |\eta'_{C\to l} - \eta_{C\to l}|\bigr)$, $\eta_{C\to l} \leftarrow \eta'_{C\to l}$.

S7. [Loop on i.] If $\delta \geq \epsilon$, set $i \leftarrow i+1$ and return to S2.

S8. [Reduce the problem.] Assign a value to each variable whose field satisfies $|p-q| \geq \psi$. (Exercise 362 has further details.) ▮

Computational experience — otherwise known as trial and error — suggests suitable parameter values. The defaults $\rho = .995$, $N_0 = 5$, $N = 1000$, $\epsilon = .01$, and $\psi = .50$ seem to provide a decent starting point for problems of modest size. They worked well, for instance, when the author first tried a random 3SAT problem with 42,000 clauses and 10,000 variables: These clauses were pseudo-satisfiable when $i = 143$ (although $\delta \approx .43$ was still rather large); then step S8 fixed the values of 8,282 variables with highly biased fields, and unit propagation gave values to 57 variables more. This process needed only about 218 megamems of calculation. The reduced problem had 1526 2-clauses and 196 3-clauses on

WalkSAT+
Baldassi
damping factor
parameter
defaults
author
3SAT
unit propagation

1464 variables (because many other variables were no longer needed); 626 steps of WalkSAT polished it off after an additional 42 kilomems. By contrast, when WalkSAT was presented with the original problem (using $p = .57$), it needed more than 31 million steps to find a solution after 3.4 gigamems of computation. author

Similarly, the author's first experience applying survey propagation to a random 3SAT problem on $n = 10^6$ variables with $m = 4.2n$ clauses was a smashing success: More than 800,000 variables were eliminated after only 32.8 gigamems of computation, and WalkSAT solved the residual clauses after 8.5 megamems more. By contrast, pure WalkSAT needed 237 gigamems to perform 2.1 billion steps.

A million-variable problem with 4,250,000 clauses proved to be more challenging. These additional 50,000 clauses put the problem well beyond WalkSAT's capability; and Algorithm S failed too, with its default parameters. However, the settings $\rho = .9999$ and $N_0 = 9$ slowed the reinforcement down satisfactorily, and produced some instructive behavior. Consider the matrix

$$\begin{pmatrix} 3988 & 3651 & 3071 & 2339 & 1741 & 1338 & 946 & 702 & 508 & 329 \\ 5649 & 5408 & 4304 & 3349 & 2541 & 2052 & 1448 & 1050 & 666 & 510 \\ 8497 & 7965 & 6386 & 4918 & 3897 & 3012 & 2248 & 1508 & 1075 & 718 \\ 11807 & 11005 & 8812 & 7019 & 5328 & 4135 & 3117 & 2171 & 1475 & 1063 \\ 15814 & 14789 & 11726 & 9134 & 7188 & 5425 & 4121 & 3024 & 2039 & 1372 \\ 20437 & 19342 & 15604 & 12183 & 9397 & 7263 & 5165 & 3791 & 2603 & 1781 \\ 26455 & 24545 & 19917 & 15807 & 12043 & 9161 & 6820 & 5019 & 3381 & 2263 \\ 33203 & 31153 & 25052 & 19644 & 15587 & 11802 & 8865 & 6309 & 4417 & 2919 \\ 39962 & 38097 & 31060 & 24826 & 18943 & 14707 & 10993 & 7924 & 5225 & 3637 \\ 40731 & 40426 & 32716 & 26561 & 20557 & 15739 & 11634 & 8327 & 5591 & 4035 \end{pmatrix},$$

which shows the distribution of $\pi_{\bar v}$ versus π_v (see Fig. 51); for example, '3988' at the upper left means that 3988 of the million variables had $\pi_{\bar v}$ between 0.0 and 0.1 and π_v between 0.9 and 1.0. This distribution, which appeared after δ had been reduced to ≈ 0.0098 by 110 iterations, is terrible — very few variables are biased in a meaningful way. Therefore another run was made with ϵ reduced to .001; but that failed to converge after 1000 iterations. Finally, with $\epsilon = .001$ and $N = 2000$, pseudo-satisfaction occurred at $i = 1373$, with the nice distribution

$$\begin{pmatrix} 406678 & 1946 & 1045 & 979 & 842 & 714 & 687 & 803 & 1298 & 167649 \\ 338 & 2 & 2 & 3 & 0 & 3 & 1 & 4 & 2 & 1289 \\ 156 & 1 & 0 & 0 & 0 & 1 & 0 & 2 & 1 & 875 \\ 118 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 743 \\ 99 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 663 \\ 62 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 3 & 810 \\ 41 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1015 \\ 55 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1139 \\ 63 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 2 & 1949 \\ 116 & 61 & 72 & 41 & 61 & 103 & 120 & 162 & 327 & 406839 \end{pmatrix}$$

(although δ was now ≈ 1!). The biases were now pronounced, yet not entirely reliable; the ψ parameter had to be raised, in order to avoid a contradiction when propagating unit literals in the reduced problem. Finally, with $\psi = .99$, more than 800,000 variables could be set successfully. A solution was obtained after 210 gigamems (including 21 megamems for WalkSAT to finish the job, but not including the time spent learning how to set the parameters for so many clauses).

Success with Algorithm S isn't guaranteed. But hey, when it works, it's sometimes the only known way to solve a particularly tough problem.

Survey propagation may be viewed as an extension of the "belief propagation" messages used in the study of Bayesian networks [see J. Pearl, *Probabilistic Reasoning in Intelligent Systems* (1988), Chapter 4]; it essentially goes beyond Boolean logic on $\{0,1\}$ to a three-valued logic on $\{0,1,*\}$. Analogous message-passing heuristics had actually been considered much earlier by H. A. Bethe and R. E. Peierls [*Proc. Royal Society of London* **A150** (1935), 552–575], and independently by R. G. Gallager [*IRE Transactions* **IT-8** (1962), 21–28]. For further information see M. Mézard and A. Montanari, *Information, Physics, and Computation* (2009), Chapters 14–22.

belief propagation
Bayesian networks
Pearl
Bethe
Peierls
Gallager
Mézard
Montanari
Preprocessing–
simplifications
data structures
restarting
inprocessing
downhill transformations
erp rule

***Preprocessing of clauses.** A SAT-solving algorithm will often run considerably faster if its input has been transformed into an equivalent but simpler set of clauses. Such transformations and simplifications typically require data structures that would be inappropriate for the main work of a solver, so they are best considered separately.

Of course we can combine a preprocessor and a solver into a single program; and "preprocessing" techniques can be applied again after new clauses have been learned, if we reach a stage where we want to clean up and start afresh. In the latter case the simplifications are called *inprocessing*. But the basic ideas are most easily explained by assuming that we just want to preprocess a given family of clauses F. Our goal is to produce nicer clauses F', which are satisfiable if and only if F is satisfiable.

We shall view preprocessing as a sequence of elementary transformations

$$F \;=\; F_0 \to F_1 \to \cdots \to F_r \;=\; F', \qquad (160)$$

where each step $F_j \to F_{j+1}$ "flows downhill" in the sense that it either (i) eliminates a variable without increasing the number of clauses, or (ii) retains all the variables but decreases the number of literals in clauses. Many different downhill transformations are known; and we can try to apply each of the gimmicks in our repertoire, in some order, until none of them lead to any further progress.

Sometimes we'll actually *solve* the given problem, by reaching an F' that is either trivially satisfiable ($\emptyset$) or trivially unsatisfiable (contains ϵ). But we probably won't be so lucky unless F was pretty easy to start with, because we're going to consider only downhill transformations that are quite simple.

Before discussing particular transformations, however, let's think about the endgame: Suppose F has n variables but F' has $n' < n$. After we've fed the clauses F' into a SAT solver and received back a solution, $x'_1 \ldots x'_{n'}$, how can we convert it to a full solution $x_1 \ldots x_n$ of the original problem F? Here's how: For every transformation $F_j \to F_{j+1}$ that eliminates a variable x_k, we shall specify an *erp rule* (so-called because it reverses the effect of *pre*processing). An erp rule for elimination is simply an assignment '$l \leftarrow E$', where l is x_k or $\bar{x}_k$, and E is a Boolean expression that involves only variables that have not been eliminated. We undo the effect of elimination by assigning to x_k the value that makes l true if and only if E is true.

For example, suppose two transformations remove x and y with the erp rules

$$\bar{x} \leftarrow \bar{y} \vee z, \qquad y \leftarrow 1.$$

To reverse these eliminations, right to left, we would set y true, then $x \leftarrow \bar{z}$.

As the preprocessor discovers how to eliminate variables, it can immediately write the corresponding erp rules to a file, so that those rules don't consume memory space. Afterwards, given a reduced solution $x'_1 \ldots x'_{n'}$, a postprocessor can read that file *in reverse order* and provide the unreduced solution $x_1 \ldots x_n$.

Transformation 1. Unit conditioning. If a unit clause '(l)' is present, we can replace F by $F\,|\,l$ and use the erp rule $l \leftarrow 1$. This elementary simplification will be carried out naturally by most solvers; but it is perhaps even more important in a preprocessor, since it often enables further transformations that the solver would not readily see. Conversely, other transformations in the preprocessor might enable unit conditionings that will continue to ripple down.

One consequence of unit conditioning is that all clauses of F' will have length two or more, unless F' is trivially unsatisfiable.

Transformation 2. Subsumption. If every literal in clause C appears also in another clause C', we can remove C'. In particular, duplicate clauses will be discarded. No erp rule is needed, because no variable goes away.

Transformation 3. Self-subsumption. If every literal in C except $\bar{x}$ appears also in another clause C', where C' contains x, we can delete x from C' because $C' \setminus x = C \diamond C'$. In other words, the fact that C *almost* subsumes C' allows us at least to *strengthen* C', without actually removing it. Again there's no erp rule. [Self-subsumption was called "the replacement principle" by J. A. Robinson in *JACM* **12** (1965), 39.]

Exercise 374 discusses data structures and algorithms by which subsumptions and self-subsumptions can be discovered with reasonable efficiency.

Transformation 4. Downhill resolution. Suppose x appears only in clauses $C_1, \ldots, C_p$ and $\bar{x}$ appears only in $C'_1, \ldots, C'_q$. We've observed (see (112)) that variable x can be eliminated if we replace those $p+q$ clauses by the pq clauses $\{C_i \diamond C'_j \mid 1 \le i \le p, 1 \le j \le q\}$. The corresponding erp rule (see exercise 367) is

$$\text{either} \quad \bar{x} \leftarrow \bigwedge_{i=1}^{p} (C_i \setminus x) \qquad \text{or} \quad x \leftarrow \bigwedge_{j=1}^{q} (C'_j \setminus \bar{x}). \tag{161}$$

Every variable can be eliminated in this way, but we might be flooded with too many clauses. We can prevent this by limiting ourselves to "downhill" cases, in which the new clauses don't outnumber the old ones. The condition $pq \le p+q$ is equivalent to $(p-1)(q-1) \le 1$, as noted above following (112); the variable is always removed in such cases. But the number of new clauses might be small even when pq is large, because of tautologies or subsumption. Furthermore, N. Eén and A. Biere wrote a fundamental paper on preprocessing [*LNCS* **3569** (2005), 61–75] that introduced important special cases in which many of the pq potential clauses can be omitted; see exercise 369. Therefore a preprocessor typically tries

postprocessor
Unit conditioning
BCP (Boolean constraint propagation), see unit
$F\,|\,l$
Subsumption
Self-subsumption
strengthening a clause
replacement principle
Robinson
data structures
Downhill resolution
variable elimination
elimination of variables
Eén
Biere

to eliminate via resolution whenever $\min(p,q) \le 10$, say, and abandons the attempt only when more than $p+q$ resolvents have been generated.

Many other transformations are possible, although the four listed above have proved to be the most effective in practice. We could, for instance, look for *failed literals*: If unit propagation leads to a contradiction when we assume that some literal l is true (namely when $F \wedge (l) \vdash_1 \epsilon$), then we're allowed to assume that l is false (because the unit clause $(\bar{l})$ is certifiable). This observation and several others related to it were exploited in the lookahead mechanisms of Algorithm Y above. But Algorithm C generally has no trouble finding failed literals all by itself, as a natural byproduct of its mechanism for resolving conflicts. Exercises 378–384 discuss other techniques that have been proposed for preprocessing.

Sometimes preprocessing turns out to be dramatically successful. For example, the anti-maximal-element clauses of exercise 228 can be proved unsatisfiable via transformations 1–4 after only about 400 megamems of work when $m = 50$. Yet Algorithm C spends 3 gigamems on that untransformed problem when m is only 14; and it needs 11 Gμ when $m = 15$, ..., failing utterly before $m = 20$.

A more typical example arises in connection with Fig. 35 above: The problem of showing that there's no 4-step path to **LIFE** involves 8725 variables, 33769 clauses, and 84041 literals, and Algorithm C requires about 6 gigamems to demonstrate that those clauses are unsatisfiable. Preprocessing needs less than 10 *mega*mems to reduce that problem to just 3263 variables, 19778 clauses, and 56552 literals; then Algorithm C can handle those with 5 Gμ of further work.

On the other hand, preprocessing might take too long, or it might produce clauses that are more difficult to deal with than the originals. It's totally useless on the *waerden* or *langford* problems. (Further examples are discussed below.)

Encoding constraints into clauses. Some problems, like *waerden*$(j,k;n)$, are inherently Boolean, and they're essentially given to us as native-born ANDs of ORs. But in most cases we can represent a combinatorial problem via clauses in many different ways, not immediately obvious, and the particular encoding that we choose can have an enormous effect on the speed with which a SAT solver is able to crank out an answer. Thus the art of problem encoding turns out to be just as important as the art of devising algorithms for satisfiability.

Our study of SAT instances has already introduced us to dozens of interesting encodings; and new applications often lead to further ideas, because Boolean algebra is so versatile. Each problem may seem at first to need its own special tricks. But we'll see that several general principles are available for guidance.

In the first place, different solvers tend to like different encodings: An encoding that's good for one algorithm might be bad for another.

Consider, for example, the *at-most-one* constraint, $y_1 + \cdots + y_p \le 1$, which arises in a great many applications. The obvious way to enforce this condition is to assert $\binom{p}{2}$ binary clauses $(\bar{y}_i \vee \bar{y}_j)$, for $1 \le i < j \le p$, so that $y_i = y_j = 1$ is forbidden; but those clauses become unwieldy when p is large. The alternative encoding in exercise 12, due to Marijn Heule, does the same job with only $3p - 6$ binary constraints when $p \ge 3$, by introducing a few auxiliary variables

failed literals
unit propagation
lookahead
anti-maximal-element clauses
waerden
langford
Encoding–
at-most-one
Heule
auxiliary variables

Langford's problem
langford$''(n)$
unit propagations
binary representation
log encoding
direct encoding
sparse encoding, see direct encoding
order encoding
Crawford
Baker

$a_1, \ldots, a_{\lfloor (p-3)/2 \rfloor}$. When we formulated Langford's problem in terms of clauses, via (12), (13), and (14) above, we therefore considered two variants called *langford*(n) and *langford*$'(n)$, where the former uses the obvious encoding of at-most-one constraints and the latter uses Heule's method. Furthermore, exercise 7.1.1–55(b) encoded at-most-one constraints in yet another way, having the same number of binary clauses but about twice as many auxiliary variables; let's give the name *langford*$''(n)$ to the clauses that we get from that scheme.

We weren't ready to discuss which of the encodings works better in practice, when we introduced *langford*(n) and *langford*$'(n)$ above, because we hadn't yet examined any SAT-solving algorithms. But now we're ready to reveal the answer; and the answer is: "It depends." Sometimes *langford*$'(n)$ wins over *langford*(n); sometimes it loses. It always seems to beat *langford*$''(n)$. Here, for example, are typical statistics, with runtimes rounded to megamems (Mμ) or kilomems (Kμ):

	variables	clauses	Algorithm D	Algorithm L	Algorithm C	
langford(9)	104	1722	23 Mμ	16 Mμ	15 Mμ	(UNSAT)
langford$'$(9)	213	801	82 Mμ	16 Mμ	21 Mμ	(UNSAT)
langford$''$(9)	335	801	139 Mμ	20 Mμ	24 Mμ	(UNSAT)
langford(13)	228	5875	71685 Mμ	45744 Mμ	295571 Mμ	(UNSAT)
langford$'$(13)	502	1857	492992 Mμ	38589 Mμ	677815 Mμ	(UNSAT)
langford$''$(13)	795	1857	950719 Mμ	46398 Mμ	792757 Mμ	(UNSAT)
langford(16)	352	11494	5 Mμ	52 Mμ	301 Kμ	(SAT)
langford$'$(16)	796	2928	12 Mμ	31 Mμ	418 Kμ	(SAT)
langford$''$(16)	1264	2928	20 Mμ	38 Mμ	510 Kμ	(SAT)
langford(64)	6016	869650	(huge)	(bigger)	35 Mμ	(SAT)
langford$'$(64)	14704	53184	(huger)	(big)	73 Mμ	(SAT)
langford$''$(64)	23488	53184	(hugest)	(biggest)	304 Mμ	(SAT)

Algorithm D prefers *langford* to *langford*$'$, because it doesn't perform unit propagations very efficiently. Algorithm L, which excels at unit propagation, likes *langford*$'$ better. Algorithm C also excels at unit propagation, but it exhibits peculiar behavior: It prefers *langford*, and on satisfiable instances it zooms in quickly to find a solution; but for some reason it runs *very* slowly on unsatisfiable instances when $n \geq 10$.

Another general principle is that short encodings — encodings with few variables and/or few clauses — are *not* necessarily better than longer encodings. For example, we often need to use Boolean variables to encode the value of a variable x that actually ranges over $d > 2$ different values, say $0 \leq x < d$. In such cases it's natural to use the binary representation $x = (x_{l-1} \ldots x_0)_2$, where $l = \lceil \lg d \rceil$, and to construct clauses based on the independent bits x_j; but that representation, known as the *log encoding*, surprisingly turns out to be a bad idea in many cases unless d is large. A *direct encoding* with d binary variables $x_0, x_1, \ldots, x_{d-1}$, where $x_j = [x = j]$, is often much better. And the *order encoding* with $d-1$ binary variables $x^1, \ldots, x^{d-1}$, where $x^j = [x \geq j]$, is often better yet; this encoding was introduced in 1994 by J. M. Crawford and A. B. Baker [*AAAI Conf.* **12** (1994), 1092–1097]. In fact, exercise 408 presents an important

application where the order encoding is the method of choice even when d is 1000 or more! The order encoding is exponentially larger than the log encoding, yet it wins in this application because it allows the SAT solver to deduce consequences rapidly via unit propagation.

Graph coloring problems illustrate this principle nicely. When we tried early in this section to color a graph with d colors, we encoded the color of each vertex with a direct representation, (15); but we could have used binary notation for those colors. And we could also have used the order encoding, even though the numerical ordering of colors is irrelevant in the problem itself. With a log encoding, exercise 391 exhibits three distinct ways to enforce the constraint that adjacent vertices have different colors. With the order encoding, exercise 395 explains that it's easy to handle graph coloring. And there also are four ways to work with the direct encoding, namely (a) to insist on one color per vertex by including the at-most-one exclusion clauses (17); or (b) to allow multivalued (multicolored) vertices by omitting those clauses; or (c) to actually *welcome* multicolored vertices, by omitting (17) and forcing each color class to be a kernel, as suggested in answer 14; or (d) to include (17) but to replace the "preclusion" clauses (16) by so-called "support" clauses as explained in exercise 399.

These eight options can be compared empirically by trying to arrange 64 colored queens on a chessboard so that no queens of the same color appear in the same row, column, or diagonal. That task is possible with 9 colors, but not with 8. By symmetry we can prespecify the colors of all queens in the top row.

encoding	colors	variables	clauses	Algorithm L	Algorithm C	
univalued	8	512	7688	3333 Mμ	9813 Mμ	(UNSAT)
multivalued	8	512	5896	1330 Mμ	11997 Mμ	(UNSAT)
kernel	8	512	6408	4196 Mμ	12601 Mμ	(UNSAT)
support	8	512	13512	16796 Mμ	20990 Mμ	(UNSAT)
log(a)	8	2376	5120	(immense)	20577 Mμ	(UNSAT)
log(b)	8	192	5848	(enormous)	15033 Mμ	(UNSAT)
log(c)	8	192	5848	(enormous)	15033 Mμ	(UNSAT)
order	8	448	6215	43615 Mμ	5122 Mμ	(UNSAT)
univalued	9	576	8928	2907 Mμ	464 Mμ	(SAT)
multivalued	9	576	6624	104 Mμ	401 Mμ	(SAT)
kernel	9	576	7200	93 Mμ	87 Mμ	(SAT)
support	9	576	15480	2103 Mμ	613 Mμ	(SAT)
log(a)	9	3168	6776	(gigantic)	1761 Mμ	(SAT)
log(b)	9	256	6776	(colossal)	1107 Mμ	(SAT)
log(c)	9	256	6584	(mammoth)	555 Mμ	(SAT)
order	9	512	7008	(monstrous)	213 Mμ	(SAT)

(Each running time shown here is the median of nine runs, made with different random seeds.) It's clear from this data that the log encodings are completely unsuitable for Algorithm L; and even the order encoding confuses that algorithm's heuristics. But Algorithm L shines over Algorithm C with respect to most of the direct encodings. On the other hand, Algorithm C loves the order encoding, especially in the difficult unsatisfiable case.

unit propagation
Graph coloring problems
at-most-one
exclusion clauses
multivalued
kernel
"preclusion" clauses
"support" clauses
colored queens
chessboard
queens
median

And that's not the end of the story. H. Tajima [M.S. thesis, Kobe University (2008)] and N. Tamura noticed that order encoding has another property, which trumps all other encodings with respect to graph coloring: Every k-clique of vertices $\{v_1, \ldots, v_k\}$ in a graph allows us to append two additional "hint clauses"

$$(\bar{v}_1^{d-k+1} \vee \cdots \vee \bar{v}_k^{d-k+1}) \wedge (v_1^{k-1} \vee \cdots \vee v_k^{k-1}) \qquad (162)$$

to the clauses for d-coloring — because some vertex of the clique must have a color $\leq d-k$, and some vertex must have a color $\geq k-1$. With these additional clauses, the running time to prove unsatisfiability of the 8-coloring problem drops drastically to just $60\,\mathrm{M}\mu$ with Algorithm L, and to only $13\,\mathrm{M}\mu$ with Algorithm C. We can even reduce it to just $2\,\mathrm{M}\mu$(!) by using that idea *twice* (see exercise 396).

The order encoding has several other nice properties, so it deserves a closer look. When we represent a value x in the range $0 \leq x < d$ by the binary variables $x^j = [x \geq j]$ for $1 \leq j < d$, we always have

$$x = x^1 + x^2 + \cdots + x^{d-1}; \qquad (163)$$

hence order encoding is often known as *unary representation.* The axiom clauses

$$(\bar{x}^{j+1} \vee x^j) \qquad \text{for } 1 \leq j < d-1 \qquad (164)$$

are always included, representing the fact that $x \geq j+1$ implies $x \geq j$ for each j; these clauses force all the 1s to the left and all the 0s to the right. When $d=2$ the unary representation reduces to a one-bit encoding equal to x itself; when $d=3$ it's a two-bit encoding with 00, 10, and 11 representing 0, 1, and 2.

We might not know all of the bits x^j of x's unary encoding while a problem is in the course of being solved. But if we do know that, say, $x^3 = 1$ and $x^7 = 0$, then we know that x belongs to the interval $[3\,.\,.\,7)$.

Suppose we know the unary representation of x. Then no calculation is necessary if we want to know the unary representation of $y = x + a$, when a is a constant, because $y^j = x^{j-a}$. Similarly, $z = a - x$ is equivalent to $z^j = \bar{x}^{a+1-j}$; and $w = \lfloor x/a \rfloor$ is equivalent to $w^j = x^{aj}$. Out-of-bounds superscripts are easy to handle in formulas such as this, because $x^i = 1$ when $i \leq 0$ and $x^i = 0$ when $i \geq d$. The special case $\bar{x} = d - 1 - x$ is obtained by left-right reflection of $\bar{x}^1 \ldots \bar{x}^{j-1}$:

$$(d-1-x)^j \;=\; (\bar{x})^j \;=\; \overline{x^{d-j}}. \qquad (165)$$

If we are using the order encoding for two independent variables x and y, with $0 \leq x, y < d$, it's similarly easy to encode the additional relation $x \leq y + a$:

$$x - y \leq a \iff x \leq y + a \iff \bigwedge_{j=\max(0,a+1)}^{\min(d-1,d+a)} \bigl(\bar{x}^j \vee y^{j-a}\bigr). \qquad (166)$$

And there are analogous ways to place bounds on the sum, $x + y$:

$$x + y \leq a \iff x \leq \bar{y} + a + 1 - d \iff \bigwedge_{j=\max(0,a+2-d)}^{\min(d-1,a+1)} \bigl(\bar{x}^j \vee \bar{y}^{a+1-j}\bigr); \qquad (167)$$

Tajima
Tamura
clique
hint clauses
unary representation
axiom clauses
ternary numbers
complementation of unary representation

$$x+y \ge a \iff \bar{x} \le y-a-1+d \iff \bigwedge_{j=\max(1,a+1-d)}^{\min(d,a)} \bigl(x^j \vee y^{a+1-j}\bigr). \quad (168)$$

In fact, exercise 405 shows that the general condition $ax+by \le c$ can be enforced with at most d binary clauses, when a, b, and c are constant. Any set of such relations, involving at most two variables per constraint, is therefore a 2SAT problem.

Relations between three or more order-encoded variables can also be handled without difficulty, as long as d isn't too large. For example, conditions such as $x+y \le z$ and $x+y \ge z$ can be expressed with $O(d \log d)$ clauses of length ≤ 3 (see exercise 407). Arbitrary linear inequalities can also be represented, in principle. But of course we shouldn't expect SAT solvers to compete with algebraic methods on problems that are inherently numerical.

Another constraint of great importance in the encoding of combinatorial problems is the relation of *lexicographic order*: Given two bit vectors $x_1 \ldots x_n$ and $y_1 \ldots y_n$, we want to encode the condition $(x_1 \ldots x_n)_2 \le (y_1 \ldots y_n)_2$ as a conjunction of clauses. Fortunately there's a nice way to do this with just $3n-2$ ternary clauses involving $n-1$ auxiliary variables $a_1, \ldots, a_{n-1}$, namely

$$\bigwedge_{k=1}^{n-1} \bigl((\bar{x}_k \vee y_k \vee \bar{a}_{k-1}) \wedge (\bar{x}_k \vee a_k \vee \bar{a}_{k-1}) \wedge (y_k \vee a_k \vee \bar{a}_{k-1})\bigr) \wedge (\bar{x}_n \vee y_n \vee \bar{a}_{n-1}), \quad (169)$$

where '$\bar{a}_0$' is omitted. For example, the clauses

$$(\bar{x}_1 \vee y_1) \wedge (\bar{x}_1 \vee a_1) \wedge (y_1 \vee a_1) \wedge (\bar{x}_2 \vee y_2 \vee \bar{a}_1) \wedge (\bar{x}_2 \vee a_2 \vee \bar{a}_1) \wedge (y_2 \vee a_2 \vee \bar{a}_1) \wedge (\bar{x}_3 \vee y_3 \vee \bar{a}_2)$$

assert that $x_1x_2x_3 \le y_1y_2y_3$. And the same formula, but with the final term $(\bar{x}_n \vee y_n \vee \bar{a}_{n-1})$ replaced by $(\bar{x}_n \vee \bar{a}_{n-1}) \wedge (y_n \vee \bar{a}_{n-1})$, works for the *strict* comparison $x_1 \ldots x_n < y_1 \ldots y_n$. These formulas arise by considering the carries that occur when $(\bar{x}_1 \ldots \bar{x}_n)_2 + (1 \text{ or } 0)$ is added to $(y_1 \ldots y_n)_2$. (See exercise 414.)

The *general* problem of encoding a constraint on the Boolean variables $x_1, \ldots, x_n$ is the question of finding a family of clauses F that are satisfiable if and only if $f(x_1, \ldots, x_n)$ is true, where f is a given Boolean function. We usually introduce auxiliary variables $a_1, \ldots, a_m$ into the clauses of F, unless f can be expressed directly with a short CNF formula; thus the encoding problem is to find a "good" family F such that we have

$$f(x_1, \ldots, x_n) = 1 \iff \exists a_1 \ldots \exists a_m \bigwedge_{C \in F} C, \quad (170)$$

where each C is a clause on the variables $\{a_1, \ldots, a_m, x_1, \ldots, x_n\}$. The variables $a_1, \ldots, a_m$ can be eliminated by resolution as in (112), at least in principle, leaving us with a CNF for f — although that CNF might be huge. (See exercise 248.)

If there's a simple circuit that computes f, we know from (24) and exercise 42 that there's an equally simple "Tseytin encoding" F, with one auxiliary variable for each gate in the circuit. For example, suppose we want to encode the condition $x_1 \ldots x_n \ne y_1 \ldots y_n$. The shortest CNF expression for this function $f(x_1, \ldots, x_n, y_1, \ldots, y_n)$ has 2^n clauses (see exercise 415); but there's a simple

2SAT
lexicographic order
comparison
carries
CNF
eliminated
circuit
Tseytin encoding
gate

circuit (Boolean chain) with just $n+1$ gates:

$$a_1 \leftarrow x_1 \oplus y_1, \quad \ldots, \quad a_n \leftarrow x_n \oplus y_n, \quad f \leftarrow a_1 \vee \cdots \vee a_n.$$

Using (24) we get the $4n$ clauses

$$\bigwedge_{j=1}^{n} \bigl((\bar{x}_j \vee y_j \vee a_j) \wedge (x_j \vee \bar{y}_j \vee a_j) \wedge (x_j \vee y_j \vee \bar{a}_j) \wedge (\bar{x}_j \vee \bar{y}_j \vee \bar{a}_j)\bigr), \qquad (171)$$

together with $(a_1 \vee \cdots \vee a_n)$, as a representation of '$x_1 \ldots x_n \neq y_1 \ldots y_n$'.

But this is overkill; D. A. Plaisted and S. Greenbaum have pointed out [*Journal of Symbolic Computation* **2** (1986), 293–304] that we can often avoid about half of the clauses in such situations. Indeed, only $2n$ of the clauses (171) are necessary (and sufficient), namely the ones involving $\bar{a}_j$:

$$\bigwedge_{j=1}^{n} \bigl((x_j \vee y_j \vee \bar{a}_j) \wedge (\bar{x}_j \vee \bar{y}_j \vee \bar{a}_j)\bigr). \qquad (172)$$

The other clauses are "blocked" (see exercise 378) and unhelpful. Thus it's a good idea to examine whether all of the clauses in a Tseytin encoding are really needed. Exercise 416 illustrates another interesting case.

An efficient encoding is possible also when f has a small BDD, and in general whenever f can be computed by a short branching program. Recall the example "Pi function" introduced in 7.1.1–(22); we observed in 7.1.2–(6) that it can be written $\bigl(((x_2 \wedge \bar{x}_4) \oplus \bar{x}_3) \wedge \bar{x}_1\bigr) \oplus x_2$. Thus it has a 12-clause Tseytin encoding

$$(x_2 \vee \bar{a}_1) \wedge (\bar{x}_4 \vee \bar{a}_1) \wedge (\bar{x}_2 \vee x_4 \vee a_1) \wedge (x_3 \vee a_1 \vee a_2) \wedge (\bar{x}_3 \vee \bar{a}_1 \vee a_2) \wedge (\bar{x}_3 \vee a_1 \vee \bar{a}_2)$$
$$\wedge (x_3 \vee \bar{a}_1 \vee \bar{a}_2) \wedge (\bar{x}_1 \vee \bar{a}_3) \wedge (a_2 \vee \bar{a}_3) \wedge (x_1 \vee \bar{a}_2 \vee a_3) \wedge (x_2 \vee a_3) \wedge (\bar{x}_2 \vee \bar{a}_3).$$

The Pi function also has a short branching program, 7.1.4–(8), namely

$$I_8 = (\bar{1}?\,7\colon 6), I_7 = (\bar{2}?\,5\colon 4), I_6 = (\bar{2}?\,0\colon 1), I_5 = (\bar{3}?\,1\colon 0),$$
$$I_4 = (\bar{3}?\,3\colon 2), I_3 = (\bar{4}?\,1\colon 0), I_2 = (\bar{4}?\,0\colon 1),$$

where the instruction '$(\bar{v}?\,l\colon h)$' means "If $x_v = 0$, go to I_l, otherwise go to I_h," except that I_0 and I_1 unconditionally produce the values 0 and 1. We can convert any such branching program into a sequence of clauses, by translating '$I_j = (\bar{v}?\,l\colon h)$' into

$$(\bar{a}_j \vee x_v \vee a_l) \;\wedge\; (\bar{a}_j \vee \bar{x}_v \vee a_h), \qquad (173)$$

where a_0 is omitted, and where any clauses containing a_1 are dropped. We also omit $\bar{a}_t$, where I_t is the first instruction; in this example $t = 8$. (These simplifications correspond to asserting the unit clauses $(\bar{a}_0) \wedge (a_1) \wedge (a_t)$.) The branching program above therefore yields ten clauses,

$$(x_1 \vee a_7) \wedge (\bar{x}_1 \vee a_6) \wedge (\bar{a}_7 \vee x_2 \vee a_5) \wedge (\bar{a}_7 \vee \bar{x}_2 \vee a_4) \wedge (\bar{a}_6 \vee x_2)$$
$$\wedge (\bar{a}_5 \vee \bar{x}_3) \wedge (\bar{a}_4 \vee x_3 \vee a_3) \wedge (\bar{a}_4 \vee \bar{x}_3 \vee a_2) \wedge (\bar{a}_3 \vee \bar{x}_4) \wedge (\bar{a}_2 \vee x_4).$$

We can readily eliminate a_6, a_5, a_3, a_2, thereby getting a six-clause equivalent

$$(x_1 \vee a_7) \wedge (\bar{x}_1 \vee x_2) \wedge (\bar{a}_7 \vee x_2 \vee \bar{x}_3) \wedge (\bar{a}_7 \vee \bar{x}_2 \vee a_4) \wedge (\bar{a}_4 \vee x_3 \vee \bar{x}_4) \wedge (\bar{a}_4 \vee \bar{x}_3 \vee x_4);$$

Boolean chain
Plaisted
Greenbaum
blocked
Tseytin encoding
BDD
branching program
Pi function
eliminate

and a preprocessor will simplify this to the four-clause CNF

$$(\bar{x}_1 \vee x_2) \wedge (x_2 \vee \bar{x}_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4), \qquad (174)$$

which appeared in exercise 7.1.1–19.

Exercise 417 explains why this translation scheme is valid. The method applies to *any* branching program whatsoever: The x variables can be tested in any order — that is, the v's need not be decreasing as in a BDD; moreover, a variable may be tested more than once.

preprocessor
Unit propagation
forcing–
CDCL solver
notation '$F \vdash_1 l$'
at-most-one constraint
circuit
BDD

Unit propagation and forcing. The effectiveness of an encoding depends largely on how well that encoding avoids bad partial assignments to the variables. If we're trying to encode a Boolean condition $f(x_1, x_2, \ldots, x_n)$, and if the tentative assignments $x_1 \leftarrow 1$ and $x_2 \leftarrow 0$ cause f to be false regardless of the values of x_3 through x_n, we'd like the solver to deduce this fact without further ado, ideally by unit propagation once x_1 and $\bar{x}_2$ have been asserted. With a CDCL solver like Algorithm C, a quickly recognized conflict means a relatively short learned clause — and that's a hallmark of progress. Even better would be a situation in which unit propagation, after asserting x_1, would already force x_2 to be true; and furthermore if unit propagation after $\bar{x}_2$ would also force $\bar{x}_1$.

Such scenarios aren't equivalent to each other. For example, consider the clauses $F = (\bar{x}_1 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$. Then, using the notation '$F \vdash_1 l$' to signify that F leads to l via unit propagation, we have $F \mid x_1 \vdash_1 x_2$, but $F \mid \bar{x}_2 \not\vdash_1 \bar{x}_1$. And with the clauses $G = (\bar{x}_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$ we have $G \mid x_1 \mid \bar{x}_2 \vdash_1 \epsilon$ (see Eq. (119)), but $G \mid x_1 \not\vdash_1 x_2$ and $G \mid \bar{x}_2 \not\vdash_1 \bar{x}_1$.

Consider now the simple at-most-one constraint on just three variables, $f(x_1, x_2, x_3) = [x_1 + x_2 + x_3 \leq 1]$. We can try to represent f by proceeding methodically using the methods suggested above, either by constructing a circuit for f or by constructing f's BDD. The first alternative (see exercise 420) yields

$$F = (x_1 \vee \bar{x}_2 \vee a_1) \wedge (\bar{x}_1 \vee x_2 \vee a_1) \wedge (x_1 \vee x_2 \vee \bar{a}_1) \wedge (\bar{x}_1 \vee \bar{x}_2) \wedge (\bar{x}_3 \vee \bar{a}_1); \qquad (175)$$

the second approach (see exercise 421) leads to a somewhat different solution,

$$G = (x_1 \vee a_4) \wedge (\bar{x}_1 \vee a_3) \wedge (\bar{a}_4 \vee \bar{x}_2 \vee a_2) \wedge (\bar{a}_3 \vee x_2 \vee a_2) \wedge (\bar{a}_3 \vee \bar{x}_2) \wedge (\bar{a}_2 \vee \bar{x}_3). \quad (176)$$

But neither of these encodings is actually very good, because $F \mid x_3 \not\vdash_1 \bar{x}_1$ and $G \mid x_3 \not\vdash_1 \bar{x}_1$. Much better is the encoding that we get from the general scheme of (18) and (19) in the case $n = 3$, $r = 1$, namely

$$S = (\bar{a}_1 \vee a_2) \wedge (\bar{x}_1 \vee a_1) \wedge (\bar{x}_2 \vee a_2) \wedge (\bar{x}_2 \vee \bar{a}_1) \wedge (\bar{x}_3 \vee \bar{a}_2), \qquad (177)$$

where a_1 and a_2 stand for s_1^1 and s_2^1; or the one obtained from (20) and (21),

$$B = (\bar{x}_3 \vee a_1) \wedge (\bar{x}_2 \vee a_1) \wedge (\bar{x}_2 \vee \bar{x}_3) \wedge (\bar{a}_1 \vee \bar{x}_1), \qquad (178)$$

where a_1 stands for b_1^2. With either (177) or (178) we have $S \mid x_i \vdash_1 \bar{x}_j$ and $B \mid x_i \vdash_1 \bar{x}_j$ by unit propagation whenever $i \neq j$. And of course the obvious encoding for this particular f is best of all, because n is so small:

$$O = (\bar{x}_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee \bar{x}_3) \wedge (\bar{x}_2 \vee \bar{x}_3). \qquad (179)$$

auxiliary variables
forcing representation
primary variables
cardinality constraint
at-most-one constraint
Heule
feedback
representation F
conjunctive prime form
prime clauses

Suppose $f(x_1,\ldots,x_n)$ is a Boolean function that's represented by a family of clauses F, possibly involving auxiliary variables $\{a_1,\ldots,a_m\}$, as in (170). We say that F is a *forcing* representation if we have

$$F\,|\,L \;\vdash\; l \qquad \text{implies} \qquad F\,|\,L \;\vdash_1\; l \tag{180}$$

whenever $L \cup l$ is a set of strictly distinct literals contained in $\{x_1,\ldots,x_n, \bar{x}_1,\ldots,\bar{x}_n\}$. In other words, if the partial assignment represented by L logically implies the truth of some other literal l, we insist that unit propagation alone should be able to deduce l from $F \mid L$. The auxiliary variables $\{a_1,\ldots,a_m\}$ are exempt from this requirement; only the potential forcings between *primary* variables $\{x_1,\ldots,x_n\}$ are supposed to be recognized easily when they occur.

(*Technical point:* If $F \mid L \vdash \epsilon$, meaning that $F \mid L$ is unsatisfiable, we implicitly have $F \mid L \vdash l$ for *all* literals l. In such a case (180) tells us that $F\,|\,L \vdash_1 l$ and $F\,|\,L \vdash_1 \bar{l}$ both hold; hence $F\,|\,L$ can then be proved unsatisfiable by unit propagation alone.)

We've seen that the clauses S and B in (177) and (178) are forcing for the constraint $[x_1 + x_2 + x_3 \le 1]$, but the clauses F and G in (175) and (176) are not. In fact, the clauses of (18) and (19) that led to (177) are *always* forcing, for the general cardinality constraint $[x_1 + \cdots + x_n \le r]$; and so are the clauses of (20) and (21) that led to (178). (See exercises 429 and 430.) Moreover, the general at-most-one constraint $[x_1 + \cdots + x_n \le 1]$ can be represented more efficiently by Heule's $3(n-2)$ binary clauses and $\lfloor (n-3)/2 \rfloor$ auxiliary variables (exercise 12), or with about $n \lg n$ binary clauses and only $\lceil \lg n \rceil$ auxiliary variables (exercise 394); both of these representations are forcing.

In general, we're glad to know as soon as possible when a variable's value has been forced by other values, because the variables of a large problem typically participate in many constraints simultaneously. If we know that x can't be 0 in constraint f, then we can often conclude that some other variable y can't be 1 in some other constraint g, if x appears in both f and g. There's lots of feedback.

On the other hand it might be worse to use a large representation F that is forcing than to use a small representation G that isn't, because additional clauses can make a SAT solver work harder. The tradeoffs are delicate, and they're difficult to predict in advance.

Every Boolean constraint $f(x_1,\ldots,x_n)$ has at least one forcing representation that involves no auxiliary variables. Indeed, it's easy to see that the *conjunctive prime form* F of f — the AND of all f's prime clauses — is forcing.

Smaller representations are also often forcing, even without auxiliaries. For example, the simple constraint $[x_1 \ge x_2 \ge \cdots \ge x_n]$ has $\binom{n}{2}$ prime clauses, namely $(x_j \vee \bar{x}_k)$ for $1 \le j < k \le n$; but only $n-1$ of those clauses, the cases when $k = j+1$ as in (164), are necessary and sufficient for forcing. Exercise 424 presents another, more-or-less random example.

In the worst case, all forcing representations of certain constraints are known to be huge, even when auxiliary variables are introduced (see exercise 428). But exercises 431–441 discuss many examples of useful and instructive forcing representations that require relatively few clauses.

We've glossed over an interesting technicality in definition (180), however: A sneaky person might actually construct a representation F that is absolutely useless in practice, even though it meets all of those criteria for forcing. For example, let $G(a_1, \ldots, a_m)$ be a family of clauses that are satisfiable — but only when the auxiliary variables a_j are set to extremely hard-to-find values. Then we might have $f(x_1) = x_1$ and $F = (x_1) \wedge G(a_1, \ldots, a_m)$(!). This defect in definition (180) was first pointed out by M. Gwynne and O. Kullmann [`arXiv:1406.7398` [cs.CC] (2014), 67 pages], who have also traced the history of the subject.

To avoid such a glitch, we implicitly assume that F is an *honest* representation of f, in the following sense: Whenever L is a set of n literals that fully characterizes a solution $x_1 \ldots x_n$ to the constraint $f(x_1, \ldots, x_n) = 1$, *the clauses* $F \mid L$ *must be easy to satisfy*, using the SLUR algorithm of exercise 444. That algorithm is efficient because it does not backtrack. All of the examples in exercises 439–444 meet this test of honesty; indeed, the test is automatically passed whenever every clause of F contains at most one negated auxiliary variable.

Some authors have suggested that a SAT solver should branch only on primary variables x_i, rather than on auxiliary variables a_j, whenever possible. But an extensive study by M. Järvisalo and I. Niemelä [*LNCS* **4741** (2007), 348–363; *J. Algorithms* **63** (2008), 90–113] has shown that such a restriction is not advisable with Algorithm C, and it might lead to a severe slowdown.

Gwynne
Kullmann
honest representation
SLUR algorithm
backtrack
negated auxiliary variable
branch
primary variables
auxiliary variables
Järvisalo
Niemelä
Symmetry breaking–
symmetries
pigeons
permutation
lexicographically ordered

Symmetry breaking. Sometimes we can achieve enormous speedup by exploiting symmetries. Consider, for example, the clauses for placing $m+1$ pigeons into m holes, (106)–(107). We've seen in Lemma B and Theorem B that Algorithm C and other resolution-related methods cannot demonstrate the unsatisfiability of those clauses without performing exponentially many steps as m grows. However, the clauses are symmetrical with respect to pigeons; independently, they're also symmetrical with respect to holes: If π is any permutation of $\{0, 1, \ldots, m\}$ and if ρ is any permutation of $\{1, 2, \ldots, m\}$, the transformation $x_{jk} \mapsto x_{(j\pi)(k\rho)}$ for $0 \le j \le m$ and $1 \le k \le m$ leaves the set of clauses (106)–(107) unchanged. Thus the pigeonhole problem has $(m+1)!\,m!$ symmetries.

We'll prove below that the symmetries on the holes allow us to assume safely that the hole-occupancy vectors are lexicographically ordered, namely that

$$x_{0k}x_{1k} \ldots x_{mk} \le x_{0(k+1)}x_{1(k+1)} \ldots x_{m(k+1)}, \quad \text{for } 1 \le k < m. \tag{181}$$

These constraints preserve satisfiability; and we know from (169) that they are readily expressed as clauses. Without the help of such additional clauses the running time of Algorithm C rises from 19 megamems for $m = 7$ to 177 Mμ for $m = 8$, and then to 3.5 gigamems and 86 Gμ for $m = 9$ and 10. But with (181), the same algorithm shows unsatisfiability for $m = 10$ after only 1 megamem; and for $m = 20$ and $m = 30$ after only 284 Mμ and 3.6 Gμ, respectively.

Even better results occur when we order the *pigeon*-occupancy vectors:

$$x_{j1}x_{j2} \ldots x_{jm} \le x_{(j+1)1}x_{(j+1)2} \ldots x_{(j+1)m}, \quad \text{for } 0 \le j < m. \tag{182}$$

With these constraints added to (106) and (107), Algorithm C polishes off the case $m = 10$ in just 69 *kilo*mems. It can even handle $m = 100$ in 133 Mμ. This

remarkable improvement was achieved by adding only m^2-m new variables and $3m^2-2m$ new clauses to the original m^2+m variables and $(m+1)+(m^3+m^2)/2$ clauses of (106) and (107). (Moreover, the reasoning that justifies (182) doesn't "cheat" by invoking the mathematical pigeonhole principle behind the scenes.)

Actually that's not all. The theory of columnwise symmetry (see exercise 498) also tells us that we're allowed to add the $\binom{m}{2}$ simple binary clauses

$$(x_{(j-1)j} \vee \bar{x}_{(j-1)k}) \quad \text{for } 1 \le j < k \le m \tag{183}$$

to (106) and (107), instead of (182). This principle is rather weak in general; but it turns out to be ideally suited to pigeons: It reduces the running time for $m = 100$ to just 21 megamems, although it needs no auxiliary variables whatsoever!

Of course the status of (106)–(107) has never been in doubt. Those clauses serve merely as training wheels because of their simplicity; they illustrate the fact that many symmetry-breaking strategies exist. Let's turn now to a more interesting problem, which has essentially the same symmetries, but with the roles of pigeons and holes played by "points" and "lines" instead. Consider a set of m points and n lines, where each line is a subset of points; we will require that no two points appear together in more than one line. (Equivalently, no two lines may intersect in more than one point.) Such a configuration may be called *quad-free*, because it is equivalent to an $m \times n$ binary matrix (x_{ij}) that contains no "quad," namely no 2×2 submatrix of 1s; element x_{ij} means that point i belongs to line j. Quad-free matrices are obviously characterized by $\binom{m}{2}\binom{n}{2}$ clauses,

$$(\bar{x}_{ij} \vee \bar{x}_{ij'} \vee \bar{x}_{i'j} \vee \bar{x}_{i'j'}), \quad \text{for } 1 \le i < i' \le m \text{ and } 1 \le j < j' \le n. \tag{184}$$

What is the maximum number of 1s in an $m \times n$ quad-free matrix? [This question, when $m = n$, was posed by K. Zarankiewicz, *Colloquium Mathematicæ* **2** (1951), 301, who also considered how to avoid more general submatrices of 1s.] Let's call that value $Z(m,n)-1$; then $Z(m,n)$ is the smallest r such that every $m \times n$ matrix with r nonzero entries contains a quad.

We've actually encountered examples of this problem before, but in a disguised form. For example (see exercise 448), a Steiner triple system on v objects exists if and only if v is odd and there is a quad-free matrix with $m = v$, $n = v(v-1)/6$, and $r = v(v-1)/2$. Other combinatorial block designs have similar characterizations.

Table 5 shows the values of $Z(m,n)$ for small cases. These values were discovered by delicate combinatorial reasoning, without computer assistance; so it's instructive to see how well a SAT solver can compete against real intelligence.

The first interesting case occurs when $m = n = 8$: One can place 24 markers on a chessboard without forming a quad, but $Z(8,8) = 25$ markers is too many. If we simply add the cardinality constraints $\sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij} \ge r$ to (184), Algorithm C will quickly find a solution when $m = n = 8$ and $r = 24$. But it bogs down when $r = 25$, requiring about 10 teramems to show unsatisfiability.

Fortunately we can take advantage of $m!\,n!$ symmetries, which permute rows and columns without affecting quads. Exercise 495 shows that those symmetries

points
lines
quad-free
binary matrix
0–1 matrix
submatrix
Zarankiewicz
avoiding submatrices
$Z(m,n)$
Steiner triple system
block designs
chessboard
cardinality constraints

Table 5

$Z(m,n)$, THE MINIMUM NUMBER OF 1S WITH (184) UNSATISFIABLE

$n =$	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
$m = 2$:	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
$m = 3$:	5	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$m = 4$:	6	8	10	11	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
$m = 5$:	7	9	11	13	15	16	18	19	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38
$m = 6$:	8	10	13	15	17	19	20	22	23	25	26	28	29	31	32	33	34	35	36	37	38	39	40	41	42	43
$m = 7$:	9	11	14	16	19	22	23	25	26	28	29	31	32	34	35	37	38	40	41	43	44	45	46	47	48	49
$m = 8$:	10	12	15	18	20	23	25	27	29	31	33	34	36	37	39	40	42	43	45	46	48	49	51	52	54	55
$m = 9$:	11	13	16	19	22	25	27	30	32	34	37	38	40	41	43	44	46	47	49	50	52	53	55	56	58	59
$m = 10$:	12	14	17	21	23	26	29	32	35	37	40	41	43	45	47	48	50	52	53	55	56	58	59	61	62	64
$m = 11$:	13	15	18	22	25	28	31	34	37	40	43	45	46	48	51	52	54	56	58	60	61	63	64	66	67	69
$m = 12$:	14	16	19	23	26	29	33	37	40	43	46	49	50	52	54	56	58	61	62	64	66	67	69	71	73	74
$m = 13$:	15	17	20	24	28	31	34	38	41	45	49	53	54	56	58	60	62	65	67	68	80	72	74	76	79	80
$m = 14$:	16	18	21	25	29	32	36	40	43	46	50	54	57	59	61	64	66	69	71	73	74	76	79	81	83	85
$m = 15$:	17	19	22	26	31	34	37	41	45	48	52	56	59	62	65	68	70	73	76	78	79	81	83	86	87	89
$m = 16$:	18	20	23	27	32	35	39	43	47	51	54	58	61	65	68	71	74	77	81	82	84	86	88	91	92	94

[*References:* R. K. Guy, in *Theory of Graphs*, Tihany 1966, edited by Erdős and Katona (Academic Press, 1968), 119–150; R. J. Nowakowski, Ph.D. thesis (Univ. of Calgary, 1978), 202.]

allow us to add the lexicographic constraints

$$x_{i1}x_{i2}\ldots x_{in} \geq x_{(i+1)1}x_{(i+1)2}\ldots x_{(i+1)n}, \quad \text{for } 1 \leq i < m; \qquad (185)$$

$$x_{1j}x_{2j}\ldots x_{mj} \geq x_{1(j+1)}x_{2(j+1)}\ldots x_{m(j+1)}, \quad \text{for } 1 \leq j < n. \qquad (186)$$

(Increasing order, with '$\leq$' in place of '$\geq$', could also have been used, but decreasing order turns out to be better; see exercise 497.) The running time to prove unsatisfiability when $r = 25$ now decreases dramatically, to only about 50 megamems. And it falls to 48 Mμ if the lexicographic constraints are shortened to consider only the leading 4 elements of a row or column, instead of testing all 8.

The constraints of (185) and (186) are useful in satisfiable problems too — not in the easy case $m = n = 8$, when they aren't necessary, but for example in the case $m = n = 13$ when $r = 52$: Then they lead Algorithm C to a solution after about 200 gigamems, while it needs more than 18 teramems to find a solution without such help. (See exercise 449.)

Satisfiability-preserving maps. Let's proceed now to the promised theory of symmetry breaking. In fact, we will do more: Symmetry is about *permutations* that preserve structural properties, but we will consider arbitrary *mappings* instead. Mappings are more general than permutations, because they needn't be invertible. If $x = x_1 \ldots x_n$ is any potential solution to a satisfiability problem, our theory is based on transformations τ that map $x \mapsto x\tau = x'_1 \ldots x'_n$, where $x\tau$ is required to be a solution whenever x is a solution.

In other words, if F is a family of clauses on n variables and if $f(x) =$ [x satisfies F], then we are interested in all mappings τ for which $f(x) \leq f(x\tau)$. Such a mapping is conventionally called an *endomorphism* of the solutions.* If an

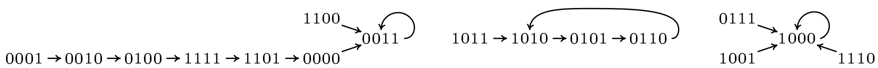
* This word is a bit of a mouthful. But it's easier to say "endomorphism" than to say "satisfiability-preserving transformation," and you can use it to impress your friends. The term "conditional symmetry" has also been used by several authors in special cases.

Guy
Erdos
Katona
Nowakowski
Satisfiability-preserving transformations–Symmetry
conditional symmetry, see endomorphism
endomorphism

endomorphism τ is actually a permutation, it's called an *automorphism*. Thus, if there are K solutions to the problem, out of $N = 2^n$ possibilities, the total number of mappings is N^N; the total number of endomorphisms is $K^K N^{N-K}$; and the total number of automorphisms is $K!\,(N-K)!$.

Notice that we don't require $f(x)$ to be exactly equal to $f(x\tau)$. An endomorphism is allowed to map a nonsolution into a solution, and only $K^K(N-K)^{N-K}$ mappings satisfy that stronger property. On the other hand, automorphisms always *do* satisfy $f(x) = f(x\tau)$; see exercise 454.

Here, for instance, is a more-or-less random mapping when $n = 4$:



(187)

Exercises 455 and 456 discuss potential endomorphisms of this mapping.

In general there will be one or more *cycles*, and every element of a cycle is the root of an oriented tree that leads to it. For example, the cycles of (187) are (0011), (1010 0101 0110), and (1000).

Several different endomorphisms $\tau_1, \tau_2, \ldots, \tau_p$ are often known. In such cases it's helpful to imagine the digraph with 2^n vertices that has arcs from each vertex x to its successors $x\tau_1, x\tau_2, \ldots, x\tau_p$. This digraph will have one or more *sink components*, which are strongly connected components Y from which there is no escape: If $x \in Y$ then $x\tau_k \in Y$ for $1 \le k \le p$. (In the special case where each τ_k is an automorphism, the sink components are traditionally called *orbits* of the automorphism group.) When $p = 1$, a sink component is the same as a cycle.

The clauses F are satisfiable if and only if $f(x) = 1$ for at least one x. Such an x will lead to at least one sink component Y, all of whose elements will satisfy $f(y) = 1$. Thus it suffices to test satisfiability by checking just one element y in every sink component Y, to see if $f(y) = 1$.

Let's consider a simple problem based on the "sweep" of an $m\times n$ matrix $X = (x_{ij})$, which is the largest diagonal sum of any $t \times t$ submatrix:

$$\operatorname{sweep}(X) \;=\; \max_{\substack{1\le i_1<i_2<\cdots<i_t\le m\\ 1\le j_1<j_2<\cdots<j_t\le n}} (x_{i_1j_1} + x_{i_2j_2} + \cdots + x_{i_tj_t}). \qquad (188)$$

When X is binary, sweep(X) is the length of the longest downward-and-rightward path that passes through its 1s. We can use satisfiability to decide whether such a matrix exists having sweep$(X) \le k$ and $\sum_{i=1}^m \sum_{j=1}^n x_{ij} \ge r$, given m, n, k, and r; suitable clauses are exhibited in exercise 460. A solution with $m = n = 10$, $k = 3$, and $r = 51$ appears at the right: It has 51 1s, but no four of them lie in a monotonic southeasterly path.

```
0000111111
0000100011
0000100111
0001101101
0111111001
1111100001
1010000011
1010000010
1110111110
1111100000
```

This problem has 2^{mn} candidate matrices X, and experiments with small m and n suggest several endomorphisms that can be applied to such candidates without increasing the sweep.

- τ_1: If $x_{ij} = 1$ and $x_{i(j+1)} = 0$, and if $x_{i'j} = 0$ for $1 \le i' < i$, we can set $x_{ij} \leftarrow 0$ and $x_{i(j+1)} \leftarrow 1$.
- τ_2: If $x_{ij} = 1$ and $x_{(i+1)j} = 0$, and if $x_{ij'} = 0$ for $1 \le j' < j$, we can set $x_{ij} \leftarrow 0$ and $x_{(i+1)j} \leftarrow 1$.

automorphism
pi as source of "random"
cycles
oriented tree
digraph
sink components
strongly connected components
orbits
sweep
trace
submatrix

auxiliary variables
lexicographically
swoop
Theory and practice
perfect matching
exact cover by pairs+
4-cycle

- τ_3: If the 2×2 submatrix in rows $\{i, i+1\}$ and columns $\{j, j+1\}$ is $\substack{11\\10}$, we can change it to $\substack{01\\11}$.

These transformations are justified in exercise 462. They're sometimes applicable for several different i and j; for instance, τ_3 could be used to change any of eight different 2×2 submatrices in the example solution. In such cases we make an arbitrary decision, by choosing (say) the lexicographically smallest possible i and j.

The clauses that encode this problem have auxiliary variables besides x_{ij}; but we can ignore the auxiliary variables when reasoning about endomorphisms.

Each of these endomorphisms either leaves X unchanged or replaces it by a lexicographically *smaller* matrix. *Therefore the sink components of* $\{\tau_1, \tau_2, \tau_3\}$ *consist of the matrices* X *that are fixed points of all three transformations.* Hence we're allowed to append additional clauses, stating that neither τ_1 nor τ_2 nor τ_3 is applicable. For instance, transformation τ_3 is ruled out by the clauses

$$\bigwedge_{i=1}^{m-1} \bigwedge_{j=1}^{n-1} (\bar{x}_{ij} \vee \bar{x}_{i(j+1)} \vee \bar{x}_{(i+1)j} \vee x_{(i+1)(j+1)}), \tag{189}$$

which state that the submatrix $\substack{11\\10}$ doesn't appear. The clauses for τ_1 and τ_2 are only a bit more complicated (see exercise 461).

These additional clauses give interesting answers in satisfiable instances, although they aren't really helpful running-time-wise. On the other hand, they're spectacularly successful when the problem is *un*satisfiable.

For example, we can show, without endomorphisms, that the case $m = n = 10$, $k = 3$, $r = 52$ is impossible, and hence that any solution for $r = 51$ is optimum; Algorithm C proves this after about 16 gigamems of work. Adding the clauses for τ_1 and τ_2, but not τ_3, increases the running time to $23\,\mathrm{G}\mu$; on the other hand the clauses for τ_3 without τ_1 or τ_2 reduce it to $6\,\mathrm{G}\mu$. When we use all three endomorphisms simultaneously, however, the running time to prove unsatisfiability goes down to just 3.5 *mega*mems, a speedup of more than 4500.

Even better is the fact that the fixed points of $\{\tau_1, \tau_2, \tau_3\}$ actually have an extremely simple form — see exercise 463 — from which we can readily determine the answer by hand, without running the machine at all! Computer experiments have helped us to guess this result; but once we've proved it, we've solved infinitely many cases in one fell swoop. Theory and practice are synergistic.

Another interesting example arises when we want to test whether or not a given graph has a *perfect matching*, which is a set of nonoverlapping edges that exactly touch each vertex. We'll discuss beautiful, efficient algorithms for this problem in Sections 7.5.1 and 7.5.5; but it's interesting to see how well a simple-minded SAT solver can compete with those methods.

Perfect matching is readily expressible as a SAT problem whose variables are called 'uv', one for each edge u — v. Variables 'uv' and 'vu' are identical. Whenever the graph contains a 4-cycle $v_0 — v_1 — v_2 — v_3 — v_0$, we might include two of its edges $\{v_0v_1, v_2v_3\}$ in the matching; but we could equally well have included $\{v_1v_2, v_3v_0\}$ instead. Thus there's an endomorphism that says, "If $v_0v_1 = v_2v_3 = 1$ (hence $v_1v_2 = v_3v_0 = 0$), set $v_0v_1 \leftarrow v_2v_3 \leftarrow 0$ and $v_1v_2 \leftarrow v_3v_0 \leftarrow 1$."

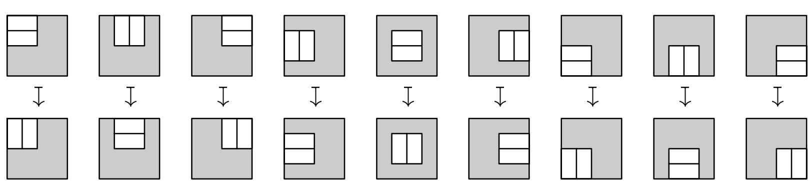
And we can carry this idea further: Let the edges be totally ordered in some arbitrary fashion, and for each edge uv consider all 4-cycles in which uv is the largest edge. In other words, we consider all cycles of the form $u — v — u' — v' — u$ in which vu', $u'v'$, $v'u$ all precede uv in the ordering. If any such cycles exist, choose one of them arbitrarily, and let τ_{uv} be one of two endomorphisms:

$$\tau^-_{uv}\text{: “If } uv = u'v' = 1\text{, set } uv \leftarrow u'v' \leftarrow 0 \text{ and } vu' \leftarrow v'u \leftarrow 1.\text{”}$$
$$\tau^+_{uv}\text{: “If } vu' = v'u = 1\text{, set } uv \leftarrow u'v' \leftarrow 1 \text{ and } vu' \leftarrow v'u \leftarrow 0.\text{”}$$

Either τ^-_{uv} or τ^+_{uv} is stipulated, for each uv. Exercise 465 proves that a perfect matching is in the sink component of any such family of endomorphisms if and only if it is fixed by all of them. Therefore we need only search for fixed points.

For example, consider the problem of covering an $m \times n$ board with dominoes. This is the problem of finding a perfect matching on the grid graph $P_m \square P_n$. The graph has mn vertices (i,j), with $m(n-1)$ “horizontal” edges h_{ij} from (i,j) to $(i,j+1)$ and $(m-1)n$ “vertical” edges v_{ij} from (i,j) to $(i+1,j)$. It has exactly $(m-1)(n-1)$ 4-cycles; and if we number the edges from left to right, no two 4-cycles have the same largest edge. Therefore we can construct $(m-1)(n-1)$ endomorphisms, in each of which we're free to decide whether to allow a particular cycle to be filled by two horizontal dominoes or by two vertical ones.

Let's stipulate that h_{ij} and $h_{(i+1)j}$ are allowed together only when $i+j$ is odd; v_{ij} and $v_{i(j+1)}$ are allowed together only when $i+j$ is even. The nine endomorphisms when $m = n = 4$ are then


. (190)

And it's not difficult to see that only *one* 4×4 domino covering is fixed by all nine. Indeed (exercise 466), the solution turns out to be unique for *all* m and n.

The famous problem of the “mutilated chessboard” asks for a domino covering when two opposite corner cells have been removed. This problem is unsatisfiable when m and n are both even, by exercise 7.1.4–213. But a SAT solver can't discover this fact quickly from the clauses alone, because there are many ways to get quite close to a solution; see the discussion following 7.1.4–(130). [S. Dantchev and S. Riis, in *FOCS* **42** (2001), 220–229, have proved in fact that every resolution refutation of these clauses requires $2^{\Omega(n)}$ steps.]

When Algorithm C is presented with mutilated boards of sizes 6×6, 8×8, 10×10, ..., 16×16, it needs respectively about 55 Kμ, 1.4 Mμ, 31 Mμ, 668 Mμ, 16.5 Gμ, and .91 Tμ (that's *tera*mems) to prove unsatisfiability. The even-odd endomorphisms typified by (190) come to our rescue, however: They narrow the search space spectacularly, reducing the respective running times to only 15 Kμ, 60 Kμ, 135 Kμ, 250 Kμ, 470 Kμ, 690 Kμ (that's *kilo*mems). They even can verify the unsatisfiability of a mutilated 256×256 domino cover after fewer than 4.2 Gμ of calculation, exhibiting a growth rate of roughly $O(n^3)$.

dominoes
grid graph
mutilated chessboard
Dantchev
Riis
resolution refutation

Endomorphisms can also speed up SAT solving in another important way:

Theorem E. *Let $p_1p_2\ldots p_n$ be any permutation of $\{1,2,\ldots,n\}$. If the Boolean function $f(x_1,x_2,\ldots,x_n)$ is satisfiable, then it has a solution such that $x_{p_1}x_{p_2}\ldots x_{p_n}$ is lexicographically less than or equal to $x'_{p_1}x'_{p_2}\ldots x'_{p_n}$ for every endomorphism of f that takes $x_1x_2\ldots x_n \mapsto x'_1x'_2\ldots x'_n$.*

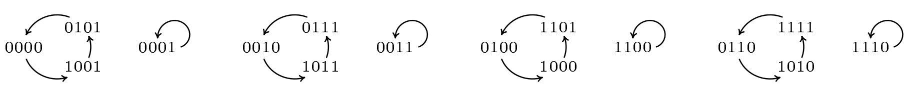
Proof. The lexicographically smallest solution of f has this property. ▮

Maybe we shouldn't call this a "theorem"; it's an obvious consequence of the fact that endomorphisms always map solutions into solutions. But it deserves to be remembered and placed on some sort of pedestal, because we will see that it has many useful applications.

Theorem E is extremely good news, at least potentially, because every Boolean function has a *huge* number of endomorphisms. (See exercise 457.) On the other hand, there's a catch: We almost never *know* any of those endomorphisms until after we've solved the problem! Still, whenever we *do* happen to know one of the zillions of nontrivial endomorphisms that exist, we're allowed to add clauses that narrow the search. There's always a "lex-leader" solution that satisfies $x_1x_2\ldots x_n \le x'_1x'_2\ldots x'_n$, if there's any solution at all.

A second difficulty that takes some of the shine away from Theorem E is the fact that most endomorphisms are too complicated to express neatly as clauses. What we really want is an endomorphism that's nice and simple, so that lexicographic ordering is equally simple.

Fortunately, such endomorphisms are often available; in fact, they're usually *automorphisms* — symmetries of the problem — defined by *signed permutations* of the variables. A signed permutation represents the operation of permuting variables and/or complementing them; for example, the signed permutation '$\bar{4}13\bar{2}$' stands for the mapping $(x_1,x_2,x_3,x_4)\mapsto(x_{\bar{4}},x_1,x_3,x_{\bar{2}})=(\bar{x}_4,x_1,x_3,\bar{x}_2)$. This operation transforms the states in a much more regular way than (187):



(191)

If σ takes the literal u into v, we write $u\sigma = v$; and in such cases σ also takes $\bar{u}$ into $\bar{v}$. Thus we always have $\bar{u}\sigma = \overline{u\sigma}$. We also write $x\sigma$ for the result of applying σ to a sequence x of literals; for example, $(x_1,x_2,x_3,x_4)\sigma = (\bar{x}_4,x_1,x_3,\bar{x}_2)$. This mapping is a symmetry or automorphism of $f(x)$ if and only if $f(x) = f(x\sigma)$ for all x. Exercises 474 and 475 discuss basic properties of such symmetries; see also exercise 7.2.1.2–20.

Notice that a signed permutation can be regarded as an unsigned permutation of the $2n$ literals $\{x_1,\ldots,x_n,\bar{x}_1,\ldots,\bar{x}_n\}$, and as such it can be written as a product of cycles. For instance, the symmetry $\bar{4}13\bar{2}$ corresponds to the cycles $(1\bar{4}2)(\bar{1}4\bar{2})(3)(\bar{3})$. We can multiply signed permutations by multiplying these cycles in the normal way, just as in Section 1.3.3.

The product $\sigma\tau$ of two symmetries σ and τ is always a symmetry. Thus in particular, if σ is any symmetry, so are its powers σ^2, σ^3, etc. We say that σ has order r if σ, σ^2, …, σ^r are distinct and σ^r is the identity. A signed permutation

lexicographically
lex-leader
signed permutations
permuting variables and/or complementing them
literals
order r

of order 1 or 2 is called a *signed involution*; this important special case arises if and only if σ is its own inverse ($\sigma^2 = 1$).

It's clearly easier to work with permutations of $2n$ literals than to work with permutations of 2^n states $x_1 \ldots x_n$. The main advantage of a signed permutation σ is that we can test whether or not σ preserves the family F of clauses in a satisfiability problem. If it does, we can be sure that σ also is an automorphism when it acts on all 2^n states. (See exercise 492.)

Let's go back to the example *waerden*$(3, 10; 97)$ that we've often discussed above. These clauses have an obvious symmetry, which takes $x_1x_2 \ldots x_{97} \mapsto x_{97}x_{96} \ldots x_1$. If we don't break this symmetry, Algorithm C typically verifies unsatisfiability after about 530 Mμ of computation. Now Theorem E tells us that we can also assert that $x_1x_2x_3 \le x_{97}x_{96}x_{95}$, say; but that symmetry-breaker doesn't really help at all, because x_1 has very little influence on x_{97}. Fortunately, however, Theorem E allows us to choose *any* permutation $p_1p_2 \ldots p_n$ on which to base lexicographic comparisons. For example, we can assert that $x_{48}x_{47}x_{46} \ldots \le x_{50}x_{51}x_{52} \ldots$ — provided that we don't *also* require $x_1x_2x_3 \ldots \le x_{97}x_{96}x_{95} \ldots$. (One fixed global ordering must be used, but the endomorphs can be arbitrary.)

Even the simple assertion that $x_{48} \le x_{50}$, which is the clause '$\overline{48}\,50$', cuts the running time down to about 410 Mμ, because this new clause combines nicely with the existing clauses 46 48 50, 48 49 50, 48 50 52 to yield the helpful binary clauses 46 50, 49 50, 50 52. If we go further and assert that $x_{48}x_{47} \le x_{50}x_{51}$, the running time improves to 345 Mμ. And the next steps $x_{48}x_{47}x_{46} \le x_{50}x_{51}x_{52}$, $\ldots$, $x_{48}x_{47}x_{46}x_{45}x_{44}x_{43} \le x_{50}x_{51}x_{52}x_{53}x_{54}x_{55}$ take us down to 290 Mμ, then 260 Mμ, 235 Mμ, 220 Mμ; we've saved more than half of the running time by exploiting a single reflection symmetry! Only 16 simple additional clauses, namely

$$\overline{48}\,50,\ \overline{48}\,a_1,\ 50\,a_1,\ \overline{47}\,51\,\bar{a}_1,\ \overline{47}\,a_2\,\bar{a}_1,\ 51\,a_2\,\bar{a}_1,\ \overline{46}\,52\,\bar{a}_2,\ \ldots,\overline{43}\,55\,\bar{a}_5$$

are needed to get this speedup, using the efficient encoding of lex order in (169).

Of course all good things come to an end, and we've now reached the point of diminishing returns: Further clauses to assert that $x_{48}x_{47} \ldots x_{42} \le x_{50}x_{51} \ldots x_{56}$ in the *waerden*$(3, 10; 97)$ problem turn out to be counterproductive.

A wonderful simplification occurs when a symmetry σ is a signed involution that has comparatively few 2-cycles. Suppose, for example, that $\sigma = 5\bar{3}\bar{2}41\bar{6}\bar{9}8\bar{7}$; in cycle form this is $(15)(\bar{1}\bar{5})(2\bar{3})(\bar{2}3)(4)(\bar{4})(6\bar{6})(7\bar{9})(\bar{7}9)(8\bar{8})$. Then the lexicographic relation $x = x_1 \ldots x_9 \le x'_1 \ldots x'_9 = x\sigma$ holds if and only if $x_1x_2x_6 \le x_5\bar{x}_3\bar{x}_6$. The reason is clear, once we look closer (see F. A. Aloul, A. Ramani, I. L. Markov, and K. A. Sakallah, *IEEE Trans.* **CAD-22** (2003), 1117–1137, §III.C): The relation $x_1 \ldots x_9 \le x'_1 \ldots x'_9$ means, in this case, "$x_1 \le x_5$; if $x_1 = x_5$ then $x_2 \le \bar{x}_3$; if $x_1 = x_5$ and $x_2 = \bar{x}_3$ then $x_3 \le \bar{x}_2$; if $x_1 = x_5$, $x_2 = \bar{x}_3$, and $x_3 = \bar{x}_2$ then $x_4 \le x_4$; if $x_1 = x_5$, $x_2 = \bar{x}_3$, $x_3 = \bar{x}_2$, and $x_4 = x_4$ then $x_5 \le x_1$; if $x_1 = x_5$, $x_2 = \bar{x}_3$, $x_3 = \bar{x}_2$, $x_4 = x_4$, and $x_5 = x_1$ then $x_6 \le \bar{x}_6$; if $x_1 = x_5$, $x_2 = \bar{x}_3$, $x_3 = \bar{x}_2$, $x_4 = x_4$, $x_5 = x_1$, and $x_6 = \bar{x}_6$ then we're done for." With this expanded description the simplifications are obvious.

In general this reasoning allows us to improve Theorem E as follows:

signed involution
inverse
waerden
reflection
lexicographic
Aloul
Ramani
Markov
Sakallah

Corollary E. *Let $p_1p_2\ldots p_n$ be any permutation of $\{1,2,\ldots,n\}$. For every signed involution σ that is a symmetry of clauses F, we can write σ in cycle form*

$$(p_{i_1} \pm p_{j_1})(\bar{p}_{i_1} \mp p_{j_1})(p_{i_2} \pm p_{j_2})(\bar{p}_{i_2} \mp p_{j_2})\ldots(p_{i_t} \pm p_{j_t})(\bar{p}_{i_t} \mp p_{j_t}) \qquad (192)$$

with $i_1 < j_1$, $i_2 < j_2$, ..., $i_t < j_t$, $i_1 < i_2 < \cdots < i_t$, and with $(\bar{p}_{i_k} \mp p_{j_k})$ omitted when $i_k = j_k$; and we're allowed to append clauses to F that assert the lexicographic relation $x_{p_{i_1}}x_{p_{i_2}}\ldots x_{p_{i_q}} \le x_{\pm p_{j_1}}x_{\pm p_{j_2}}\ldots x_{\pm p_{j_q}}$, where $q = t$ or q is the smallest k with $i_k = j_k$. ▮

In the common case when σ is an ordinary signless involution, all of the signs can be eliminated here; we simply assert that $x_{p_{i_1}}\ldots x_{p_{i_t}} \le x_{p_{j_1}}\ldots x_{p_{j_t}}$.

This involution principle justifies all of the symmetry-breaking techniques that we used above in the pigeonhole and quad-free matrix problems. See, for example, the details discussed in exercise 495.

The idea of breaking symmetry by appending clauses was pioneered by J.-F. Puget [*LNCS* **689** (1993), 350–361], then by J. Crawford, M. Ginsberg, E. Luks, and A. Roy [*Int. Conf. Knowledge Representation and Reasoning* **5** (1998), 148–159], who considered unsigned permutations only. They also attempted to discover symmetries algorithmically from the clauses that were given as input. Experience has shown, however, that useful symmetries can almost always be better supplied by a person who understands the structure of the underlying problem.

Indeed, symmetries are often "semantic" rather than "syntactic." That is, they are symmetries of the underlying Boolean function, but not of the clauses themselves. In the Zarankiewicz problem about quad-free matrices, for example, we appended efficient cardinality clauses to ensure that $\sum x_{ij} \ge r$; that condition is symmetric under row and column swaps, but the clauses are not.

In this connection it may also be helpful to mention the *monkey wrench principle*: All of the techniques by which we've proved quickly that the pigeonhole clauses are unsatisfiable would have been useless if there had been one more clause such as $(x_{01} \vee x_{11} \vee \bar{x}_{22})$; that clause would have destroyed the symmetry!

We conclude that we're allowed to *remove* clauses from F until reaching a subset of clauses F_0 for which symmetry-breakers S can be added. If $F = F_0 \cup F_1$, and if F_0 is satisfiable $\iff$ $F_0 \cup S$ is satisfiable, then $F_0 \cup S \vdash \epsilon \implies F \vdash \epsilon$.

One hundred test cases. And now — ta da! — let's get to the climax of this long story, by looking at how our SAT solvers perform when presented with 100 moderately challenging instances of the satisfiability problem. The 100 sets of clauses summarized on the next two pages come from a wide variety of different applications, many of which were discussed near the beginning of this section, while others appear in the exercises below.

Every test case has a code name, consisting of a letter and a digit. Table 6 characterizes each problem and also shows exactly how many variables, clauses, and total literals are involved. For example, the description of problem A1 ends with '2043|24772|55195|U'; this means that A1 consists of 24772 clauses on 2043 variables, having 55195 literals altogether, and those clauses are unsatisfiable. Furthermore, since '24772' is underlined, all of A1's clauses have length 3 or less.

Puget
Crawford
Ginsberg
Luks
Roy
Zarankiewicz problem
quad-free
cardinality clauses
monkey wrench principle
pigeonhole clauses
test cases–

Table 6

CAPSULE SUMMARIES OF THE HUNDRED TEST CASES

test cases, capsule summaries+ many items are indexed here but they don't show in margin!

A1. Find $x = x_1x_2\ldots x_{99}$ with $\nu x = 27$ and no three equally spaced 1s. (See exercise 31.) 2043|24772|55195|U

A2. Like A1, but $x_1x_2\ldots x_{100}$. 2071|25197|56147|S

B1. Cover a mutilated 10×10 board with 49 dominoes, *without* using extra clauses to break symmetry. 176|572|1300|U

B2. Like B1, but a 12×12 board with 71 dominoes. 260|856|1948|U

C1. Find an 8-step Boolean chain that computes $(z_2z_1z_0)_2 = x_1+x_2+x_3+x_4$. (See exercise 479(a).) 384|16944|66336|U

C2. Find a 7-step Boolean chain that computes the modified full adder functions z_1, z_2, z_3 in exercise 481(b). 469|26637|100063|U

C3. Like C2, but with 8 steps. 572|33675|134868|S

C4. Find a 9-step Boolean chain that computes z_l and z_r in the mod-3 addition problem of exercise 480(b). 678|45098|183834|S

C5. Connect `A` to `A`, ..., `J` to `J` in Dudeney's puzzle of exercise 392, (*iv*). 1980|22518|70356|S

C6. Like C5, but move the `J` in row 8 from column 4 to column 5. 1980|22518|70356|U

C7. Given binary strings s_1, ..., s_{50} of length 200, randomly generated at distances $\le r_j$ from some string x, find x (see exercise 502). 65719|577368|1659623|S

C8. Given binary strings s_1, ..., s_{40} of length 500, inspired by biological data, find a string at distance ≤ 42 from each of them. 123540|909120|2569360|U

C9. Like C8, but at distance ≤ 43. 124100|926200|2620160|S

D1. Satisfy *factor_fifo*(18, 19, 111111111111). (See exercise 41.) 1940|6374|16498|U

D2. Like D1, but *factor_lifo*. 1940|6374|16498|U

D3. Like D1, but (19, 19, 111111111111). 2052|6745|17461|S

D4. Like D2, but (19, 19, 111111111111). 2052|6745|17461|S

D5. Solve $(x_1\ldots x_9)_2\times(y_1\ldots y_9)_2 \ne (x_1\ldots x_9)_2\times'(y_1\ldots y_9)_2$, with two copies of the *same* Dadda multiplication circuit. 864|2791|7236|U

E0. Find an Erdős discrepancy pattern $x_1\ldots x_{500}$ (see exercise 482). 1603|9157|27469|S

E1. Like E0, but $x_1\ldots x_{750}$. 2556|14949|44845|S

E2. Like E0, but $x_1\ldots x_{1000}$. 3546|21035|63103|S

F1. Satisfy *fsnark*(99). (See exercise 176.) 1782|4161|8913|U

F2. Like F1, but without the clauses $(\bar e_{1,3}\vee \bar f_{99,3})\wedge(\bar f_{1,1}\vee \bar e_{2,1})$. 1782|4159|8909|S

G1. Win Late Binding Solitaire with the "most difficult winnable deal" in answer 486. 1242|22617|65593|S

G2. Like G1, but with the most difficult *unwinnable* deal. 1242|22612|65588|U

G3. Find a test pattern for the fault "B_{43}^{43} stuck at 0" in *prod*(16, 32). 3498|11337|29097|S

G4. Like G3, but for the fault "$D_{34}^{13,9}$ stuck at 0." 3502|11349|29127|S

G5. Find a 7×15 array X_0 leading to X_3 = **LIFE** as in Fig. 35, having at most 38 live cells. 7150|28508|71873|U

G6. Like G5, but at most 39 live cells. 7152|28536|71956|S

G7. Like G5, but X_4 = **LIFE** and X_0 can be arbitrary. 8725|33769|84041|U

G8. Find a configuration in the Game of Life that proves $f^*(7,7) = 28$ (see exercise 83). 97909|401836|1020174|S

K0. Color the 8×8 queen graph with 8 colors, using the direct encoding (15) and (16), also forcing the colors of all vertices in the top row. 512|5896|12168|U

K1. Like K0, but with the exclusion clauses (17) also. 512|7688|15752|U

K2. Like K1, but with kernel clauses instead of (17) (see answer 14). 512|6408|24328|U

K3. Like K1, but with support clauses instead of (16) (see exercise 399). 512|13512|97288|U

K4. Like K1, but using the order encoding for colors. 448|6215|21159|U

K5. Like K4, but with the hint clauses (162) appended. 448|6299|21663|U

K6. Like K5, but with double clique hints (exercise 396). 896|8559|27927|U

K7. Like K1, but with the log encoding of exercise 391(a). 2376|5120|15312|U

K8. Like K1, but with the log encoding of exercise 391(b). 192|5848|34968|U

L1. Satisfy *langford*(10). 130|2437|5204|U

L2. Satisfy *langford'*(10). 273|1020|2370|U

L3. Satisfy *langford*(13). 228|5875|12356|U

L4. Satisfy *langford'*(13). 502|1857|4320|U

L5. Satisfy *langford*(32). 1472|102922|210068|S

L6. Satisfy *langford'*(32). 3512|12768|29760|S

L7. Satisfy *langford*(64). 6016|869650|1756964|S

L8. Satisfy *langford'*(64). 14704|53184|124032|S

M1. Color the McGregor graph of order 10 (Fig. 33) with 4 colors, using one color at most 6 times, via the cardinality constraints (18) and (19). 1064|2752|6244|U

M2. Like M1, but via (20) and (21). 814|2502|5744|U

M3. Like M1, but at most 7 times. 1161|2944|6726|S

M4. Like M2, but at most 7 times. 864|2647|6226|S

M5. Like M4, but order 16 and at most 11 times. 2256|7801|18756|U

M6. Like M5, but at most 12 times. 2288|8080|19564|S

M7. Color the McGregor graph of order 9 with 4 colors, and with at least 18 regions doubly colored (see exercise 19). 952|4539|13875|S

M8. Like M7, but with at least 19 regions. 952|4540|13877|U

N1. Place 100 nonattacking queens on a 100×100 board. 10000|1151800|2313400|S

O1. Solve a random open shop scheduling problem with 8 machines and 8 jobs, in 1058 units of time. 50846|557823|1621693|U

O2. Like O1, but in 1059 units. 50901|558534|1623771|S

P0. Satisfy (99), (100), and (101) for $m = 20$, thereby exhibiting a poset of size 20 with no maximal element. 400|7260|22080|U

P1. Like P0, but with $m = 14$ and using only the clauses of exercise 228. 196|847|2667|U

P2. Like P0, but with $m = 12$ and using only the clauses of exercise 229. 144|530|1674|U

P3. Like P2, but omitting the clause $(\bar{x}_{31} \vee \bar{x}_{16} \vee x_{36})$. 144|529|1671|S

P4. Like P3, but with $m = 20$. 400|2509|7827|S

Q0. Like K0, but with 9 colors. 576|6624|13688|S

Q1. Like K1, but with 9 colors. 576|8928|18296|S

Q2. Like K2, but with 9 colors. 576|7200|27368|S

Q3. Like K3, but with 9 colors. 576|15480|123128|S

Q4. Like K4, but with 9 colors. 512|7008|24200|S

Q5. Like K5, but with 9 colors. 512|7092|24704|S

Q6. Like K6, but with 9 colors. 1024|9672|31864|S

Q7. Like K7, but with 9 colors. 3168|6776|20800|S

Q8. Like K8, but with 9 colors. 256|6776|52832|S

Q9. Like Q8, but with the log encoding of exercise 391(c). 256|6584|42256|S

R1. Satisfy $\mathit{rand}(3, 1061, 250, 314159)$. 250|1061|3183|S

R2. Satisfy $\mathit{rand}(3, 1062, 250, 314159)$. 250|1062|3186|U

S1. Find a 4-term disjunctive normal form on $\{x_1, \ldots, x_{20}\}$ that differs from (27) but agrees with it at 108 random training points. 356|4229|16596|S

S2. Like S1, but at 109 points. 360|4310|16760|U

S3. Find a sorting network on nine elements that begins with the comparators [1:6][2:7][3:8][4:9] and finishes in five more parallel rounds. (See exercise 64.) 5175|85768|255421|U

S4. Like S3, but in six more rounds. 6444|107800|326164|S

T1. Find a 24×100 tatami tiling that spells 'TATAMI' as in exercise 118. 2874|10527|26112|S

T2. Like T1, but 24×106 and the 'I' should have serifs. 3048|11177|27724|U

T3. Solve the TAOCP problem of exercise 389 with only 4 knight moves. 3752|12069|27548|U

T4. Like T3, but with 5 knight moves. 3756|12086|27598|S

T5. Find the pixel in row 5, column 18 of Fig. 37(c), the lexicographically last solution to the Cheshire Tom problem. 8837|39954|100314|S

T6. Like T5, but column 19. 8837|39955|100315|U

T7. Solve the run-count extension of the Cheshire Tom problem (see exercise 117). 25734|65670|167263|S

T8. Like T7, but find a solution that differs from Fig. 36. 25734|65671|167749|U

W1. Satisfy $\mathit{waerden}(3, 10; 97)$. 97|2779|11662|U

W2. Satisfy $\mathit{waerden}(3, 13; 159)$. 159|7216|31398|S

W3. Satisfy $\mathit{waerden}(5, 5; 177)$. 177|7656|38280|S

W4. Satisfy $\mathit{waerden}(5, 5; 178)$. 178|7744|38720|U

X1. Prove that the "taking turns" protocol (43) gives mutual exclusion for at least 100 steps. 1010|3612|10614|U

X2. Prove that assertions Φ for the four-bit protocol of exercise 101, analogous to (50), are invariant. 129|354|926|U

X3. Prove that Bob won't starve in 36 steps, assuming the Φ of X2. 1652|10552|28971|U

X4. Prove that there's a simple 36-step path with the four-bit protocol, assuming the Φ of X2. 22199|50264|130404|S

X5. Like X4, but 37 steps. 23388|52822|137034|U

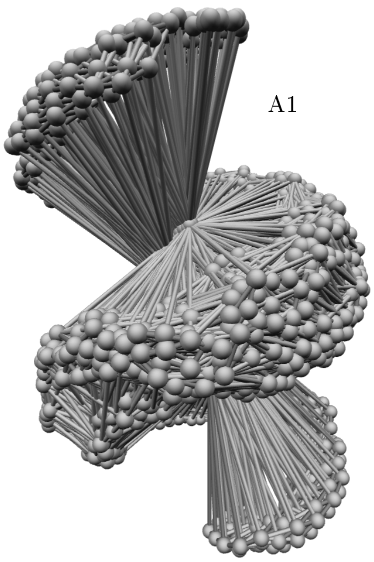
X6. Like X1, but with Peterson's protocol (49) instead of (43). 2218|8020|23222|U

X7. Prove that there's a simple 54-step path with protocol (49). 26450|56312|147572|S

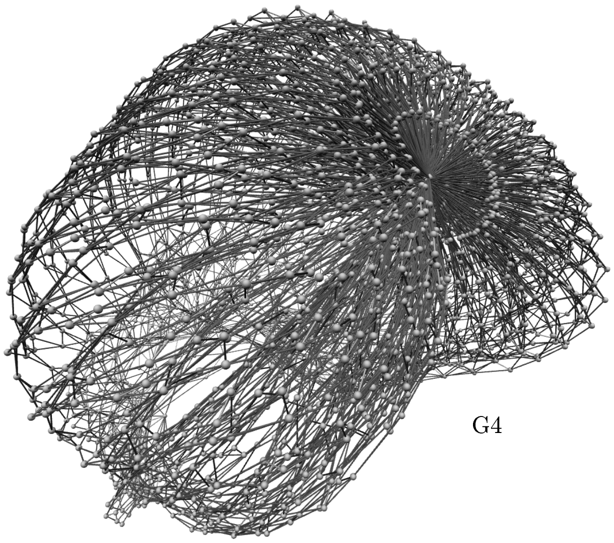
X8. Like X7, but 55 steps. 27407|58317|152807|U

menagerie+
graph layout++
geek art+
visualizations++
3D visualizations++
variable interaction graphs++

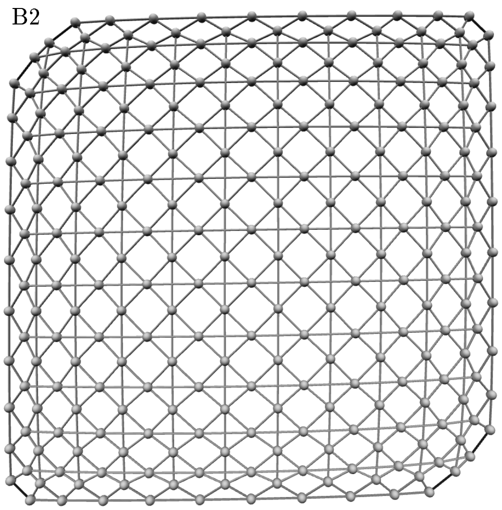
A1



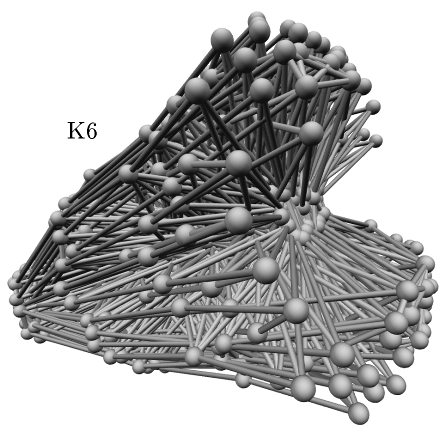
G4



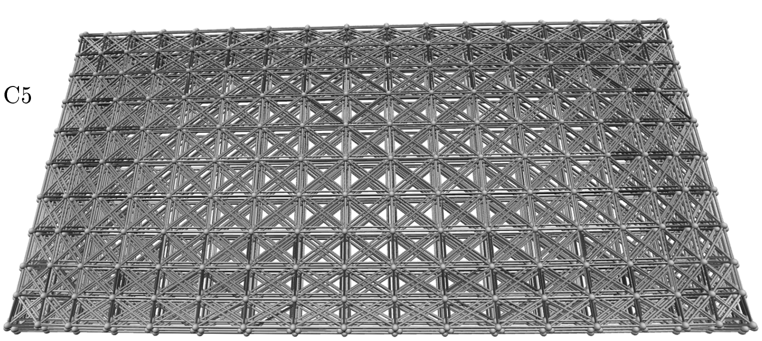
B2



K6

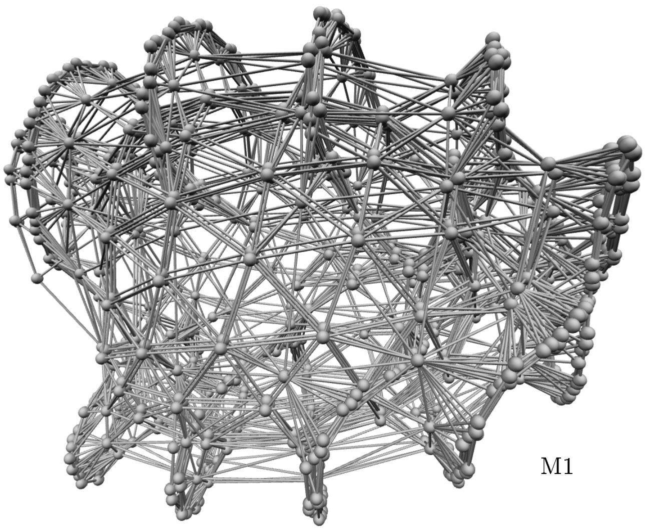


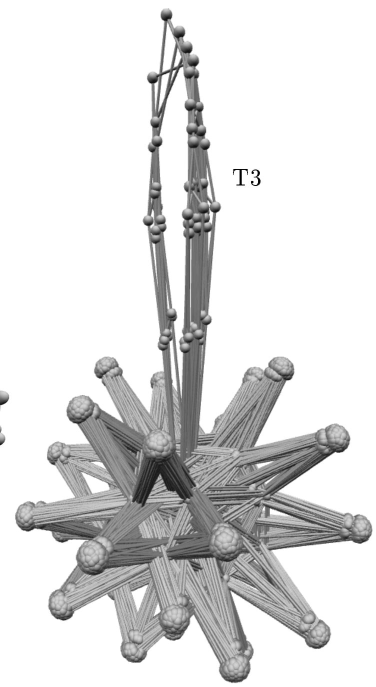
C5

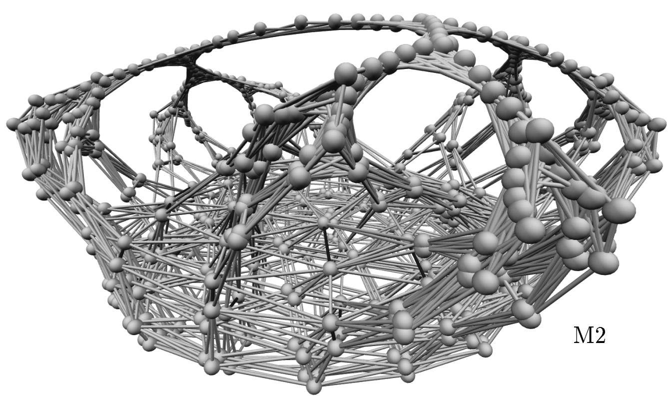


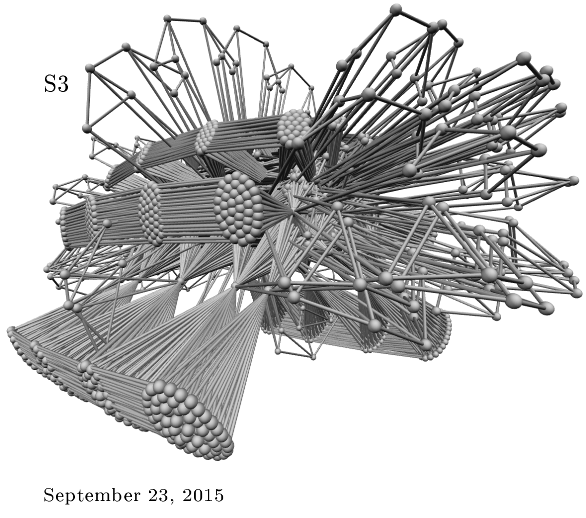
Sinz

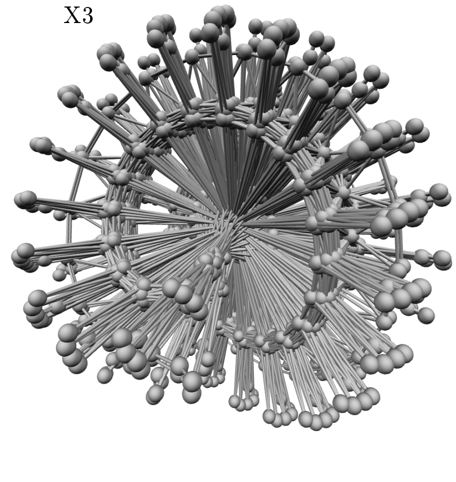
Fig. 52. The clauses of these test cases bind the variables together in significantly different ways. (Illustrations by Carsten Sinz.)











Sinz
SATexamples.tgz
Knuth
website
Knuth
lookahead algorithm
clause-learning algorithm

Of course we can't distinguish hard problems from easy ones by simply counting variables, clauses, and literals. The great versatility with which clauses can capture logical relationships means that different sets of clauses can lead to wildly different phenomena. Some of this immense variety is indicated in Fig. 52, which depicts ten instructive "variable interaction graphs." Each variable is represented by a ball, and two variables are linked when they appear together in at least one clause. (Some edges are darker than others; see exercise 506. For further examples of such 3D visualizations, presented also in color, see Carsten Sinz, *Journal of Automated Reasoning* **39** (2007), 219–243.)

A single SAT solver cannot be expected to excel on all of the many species of problems. Furthermore, nearly all of the 100 instances in Table 6 are well beyond the capabilities of the simple algorithms that we began with: Algorithms A, B, and D are unable to crack *any* of those test cases without needing more than fifty gigamems of computation, except for the simplest examples — L1, L2, L5, P3, P4, and X2. Algorithm L, the souped-up refinement of Algorithm D, also has a lot of difficulty with most of them. On the other hand, Algorithm C does remarkably well. It polishes off 79 of the given problems in fewer than *ten* $\mathrm{G}\mu$.

Thus the test cases of Table 6 are tough, yet they're within reach. Almost all of them can be solved in say two minutes, at most, with methods known today.

Complete details can be found in the file `SATexamples.tgz` on the author's website, together with many related problems both large and small.

Exactly 50 of these 100 cases are satisfiable. So we're naturally led to wonder whether Algorithm W ("WalkSAT") will handle such cases well. The answer is that Algorithm W sometimes succeeds brilliantly — especially on problems C7, C9, L5, L7, M3, M4, M6, P3, P4, Q0, Q1, R1, S1, where it typically outperforms all the other methods we've discussed. In particular it solved S1 in just $1\,\mathrm{M}\mu$, in the author's tests, compared to $25\,\mathrm{M}\mu$ by the next best method, Algorithm C; it won by $15\,\mathrm{M}\mu$ versus Algorithm C's $83\,\mathrm{M}\mu$ on M3, by $83\,\mathrm{M}\mu$ versus Algorithm L's $104\,\mathrm{M}\mu$ on Q0, by $95\,\mathrm{M}\mu$ versus Algorithm C's $464\,\mathrm{M}\mu$ on Q1, and by a whopping $104\,\mathrm{M}\mu$ versus Algorithm C's $7036\,\mathrm{M}\mu$ on C7. That was a surprise. WalkSAT also was reasonably competitive on problem N1. But in all other cases it was nowhere near the method of choice. Therefore we'll consider only Algorithms L and C in the remainder of this discussion.*

When does a lookahead algorithm like Algorithm L outperform a clause-learning algorithm like Algorithm C? Figure 53 shows how they compare to each other on our 100 test cases: Each problem is plotted with Algorithm C's running time on the vertical axis and Algorithm L's on the horizontal axis. Thus Algorithm L is the winner for problems that appear above the dotted line. (This dotted line is "wavy" because times aren't drawn to scale: The kth fastest running time is shown as k units from the left of the page or from the bottom.)

* There actually are *two* variants of Algorithm L, because the alternative heuristics of exercise 143 must be used for looking ahead when clauses of length 4 or more are present. We could use exercise 143 even when given all-ternary clauses; but experience shows that we'd tend to lose a factor of 2 or more by doing so. Our references to Algorithm L therefore implicitly assume that exercise 143 is being applied only when necessary.

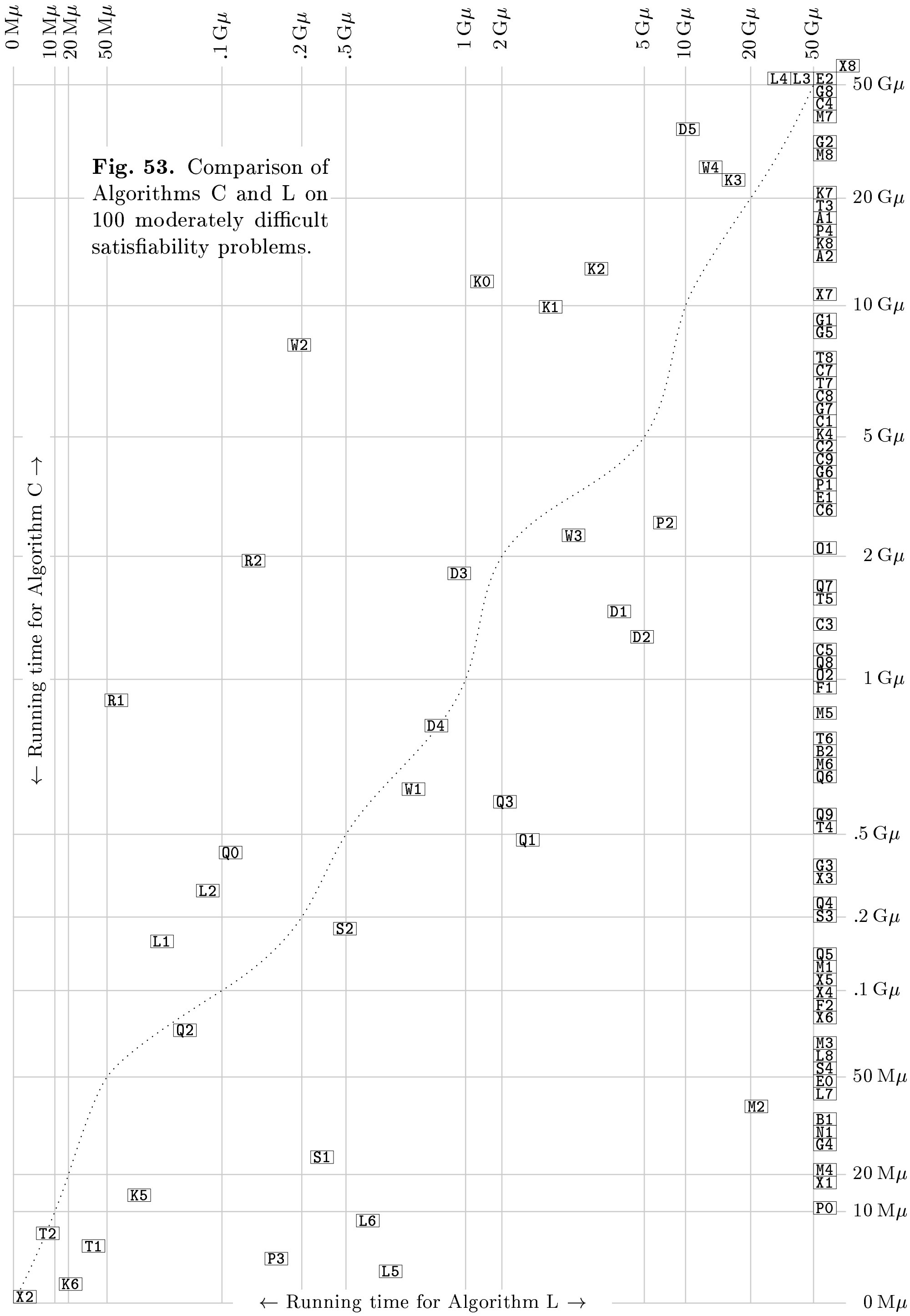


Fig. 53. Comparison of Algorithms C and L on 100 moderately difficult satisfiability problems.

All of these experiments were aborted after $50\,\mathrm{G}\mu$, if necessary, since many of these problems could potentially take centuries before running to completion. Thus the test cases for which Algorithm L timed out appear at the right edge of Fig. 53, and the tough cases for Algorithm C appear at the top. Only E2 and X8 were too hard for both algorithms to handle within the specified cutoff time.

Algorithm L is deterministic: It uses no random variables. However, a slight change (see exercise 505) will randomize it, because the inputs can be shuffled as they are in Algorithm C; and we might as well assume that this change has been made. Then both Algorithms L and C have variable running times. They will find solutions or prove unsatisfiability more quickly on some runs than on others, as we've already seen for Algorithm C in Fig. 49.

To compensate for this variability, each of the runtimes reported in Fig. 53 is the *median* of nine independent trials. Figure 54 shows all 9×100 of the empirical running times obtained with Algorithm C, sorted by their median values. We can see that many of the problems have near-constant behavior; indeed, the ratio max/min was less than 2 in 38 of the cases. But 10 cases turned out to be highly erratic in these experiments, with max/min >100; problem P4 was actually solved once after only 323 *kilo*mems, while another run lasted 339 *giga*mems!

One might expect satisfiable problems, such as P4, to benefit more from lucky guesses than unsatisfiable problems do; and these experiments strongly support that hypothesis: Of the 21 problems with max/min >30, all but P0 are satisfiable, and all 32 of the problems with max/min <1.7 are unsatisfiable. One might also expect the mean running time (the arithmetic average) to exceed the median running time, in problems like this — because bad luck can be significantly bad, though hopefully rare. Yet the mean is actually *smaller* than the median in 30 cases, about equally distributed between satisfiable and unsatisfiable.

The median is a nice measure because it is meaningful even in the presence of occasional timeouts. It's also fair, because we are able to achieve the median time, or better, more often than not.

We should point out that input/output has been excluded from these time comparisons. Each satisfiability problem is supposed to appear within a computer's memory as a simple list of clauses, after which the counting of mems actually begins. We include the cost of initializing the data structures and solving the problem, but then we stop counting before actually outputting a solution.

Some of the test cases in Table 6 and Fig. 53 represent different encodings of the same problem. For example, problems K0–K8 all demonstrate that the 8×8 queen graph can't be colored with 8 colors. Similarly, problems Q0–Q9 all show that 9 colors will suffice. We've already discussed these examples above when considering alternative encodings; and we noted that the best solutions, K6 and Q5, are obtained with an extended order encoding and with Algorithm C. Therefore the fact that Algorithm L beats Algorithm C on problems K0, K1, K2, and K3 is somewhat irrelevant; those problems won't occur in practice.

Problems L5 and L6 compare different ways to handle the at-most-one constraint. L6 is slightly better for Algorithm L, but Algorithm C prefers L5. Similarly, M1 and M2 compare different ways to deal with a more general

deterministic
median
mean running time
average
timeouts
input/output
encodings
queen graph
order encoding
at-most-one

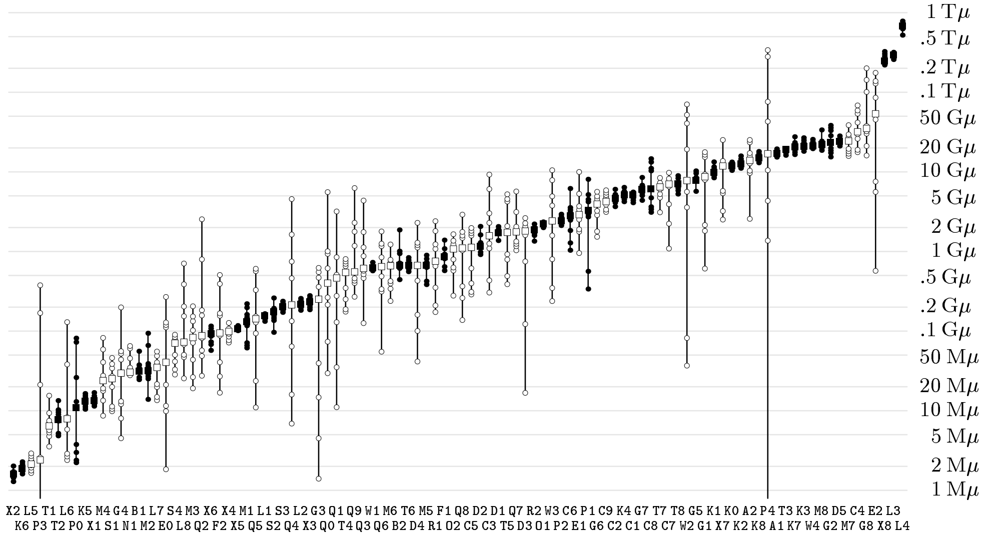


Fig. 54. Nine random running times of Algorithm C, sorted by their medians. (Unsatisfiable cases have solid dots or squares; satisfiable cases are hollow.)

cardinality constraint
waerden
Langford
CDCL solver
Treengeling
dancing links
parallel computation
CPU: Central Processing Unit (one computer th
miter
gates

cardinality constraint. Here M2 turns out to be better, although both are quite easy for Algorithm C and difficult for Algorithm L.

We've already noted that Algorithm L shines with respect to random problems such as R1 and R2, and it dominates all competitors even more when unsatisfiable random 3SAT problems get even bigger. Lookahead methods are also successful in *waerden* problems like W1–W4.

Unsatisfiable Langford problems such as L3 and L4 are definitely *bêtes noires* for Algorithm C, although not so bad for Algorithm L. Even the world's fastest CDCL solver, "Treengeling," was unable to refute the clauses of *langford*(17) in 2013 until it had learned 26.7 billion clauses; this process took more than a week, using a cluster of 24 computers working together. By contrast, the dancing links method of Section 7.2.2.1 was able to prove unsatisfiability after fewer than 7.2 Tμ of computation — that's about 90 minutes on a single vintage-2013 CPU.

We've now discussed every case where Algorithm L trumps Algorithm C, *except* for D5; and D5 is actually somewhat scandalous! It's an inherently simple problem that hardware designers call a "miter": Imagine two *identical* circuits that compute some function $f(x_1, \ldots, x_n)$, one with gates $g_1, \ldots, g_m$ and another with corresponding gates $g'_1, \ldots, g'_m$, all represented as in (24). The problem is to find $x_1 \ldots x_n$ for which the final results g_m and g'_m aren't equal. It's obviously unsatisfiable. Furthermore, there's an obvious way to refute it, by successively learning the clauses $(\bar{g}_1 \vee g'_1)$, $(\bar{g}'_1 \vee g_1)$, $(\bar{g}_2 \vee g'_2)$, $(\bar{g}'_2 \vee g_2)$, etc. In theory, therefore, Algorithm C will almost surely finish in polynomial time (see exercise 386). But in practice, the algorithm won't discover those clauses without quite a lot of flailing around, unless special-purpose techniques are introduced to help it discover isomorphic gates.

Thus Algorithm C does have an Achilles heel or two. On the other hand, it is the clear method of choice in the vast majority of our test cases, and we can expect it to be the major workhorse for most of the satisfiability problems that we encounter in daily work. Therefore it behooves us to understand its behavior in some detail, not just to look at its total cost as measured in mems.

empirical performance measurements++

Table 7

ALGORITHM C'S EMPIRICAL BEHAVIOR ON THE HUNDRED TEST CASES

name	runtime	bytes	cells	nodes	learned	of size	triv	disc	sub	flushes	sat?
X2	0+2 Mμ	57 K	9 K	2 K	1 K	32.0 → 12.0	50%	6%	1%	30	U
K6	0+2 Mμ	314 K	46 K	1 K	0 K	15.8 → 11.8	22%	4%	3%	6	U
L5	1+1 Mμ	1841 K	210 K	0 K	0 K	146.1 → 38.4	51%	23%	0%	0	S
P3	0+2 Mμ	96 K	19 K	2 K	1 K	18.4 → 12.6	4%	11%	1%	45	S
T1	0+6 Mμ	541 K	35 K	3 K	1 K	7.4 → 6.8	3%	2%	6%	9	S
T2	0+7 Mμ	574 K	37 K	4 K	1 K	7.2 → 6.8	1%	2%	4%	6	U
L6	0+8 Mμ	672 K	39 K	1 K	0 K	195.9 → 67.8	86%	0%	0%	0	S
P0	0+11 Mμ	376 K	81 K	8 K	4 K	17.8 → 14.7	3%	10%	10%	28	U
K5	0+13 Mμ	294 K	55 K	3 K	2 K	18.6 → 12.4	33%	1%	1%	14	U
X1	0+13 Mμ	284 K	38 K	29 K	4 K	6.3 → 5.8	0%	3%	8%	53	U
M4	0+24 Mμ	308 K	47 K	6 K	4 K	20.5 → 16.3	14%	2%	1%	3	S
S1	0+25 Mμ	366 K	72 K	9 K	4 K	34.0 → 26.7	22%	4%	1%	14	S
G4	0+29 Mμ	759 K	76 K	3 K	2 K	37.1 → 24.2	26%	0%	0%	1	S
N1	16+14 Mμ	19644 K	2314 K	41 K	0 K	629.3 → 291.7	44%	6%	0%	15	S
B1	0+31 Mμ	251 K	55 K	10 K	7 K	13.5 → 11.3	3%	5%	4%	14	U
M2	0+32 Mμ	326 K	53 K	7 K	5 K	18.2 → 12.8	20%	1%	1%	6	U
L7	12+23 Mμ	14695 K	1758 K	2 K	1 K	411.2 → 107.6	66%	4%	0%	0	S
E0	0+40 Mμ	571 K	95 K	5 K	3 K	30.2 → 19.3	14%	11%	0%	6	S
S4	1+69 Mμ	3291 K	600 K	6 K	2 K	17.2 → 12.6	19%	1%	1%	8	S
L8	1+72 Mμ	3047 K	224 K	3 K	2 K	547.9 → 169.1	87%	0%	0%	0	S
M3	0+83 Mμ	493 K	84 K	13 K	9 K	28.4 → 19.2	31%	0%	1%	1	S
Q2	0+87 Mμ	885 K	190 K	11 K	8 K	61.7 → 45.8	36%	0%	0%	11	S
X6	0+93 Mμ	775 K	122 K	86 K	17 K	13.5 → 11.4	0%	3%	3%	32	U
F2	0+95 Mμ	714 K	118 K	42 K	22 K	14.3 → 13.1	0%	2%	4%	5	S
X4	1+98 Mμ	3560 K	158 K	24 K	3 K	16.2 → 11.4	9%	2%	3%	623	S
X5	1+106 Mμ	3747 K	166 K	23 K	3 K	16.5 → 11.0	11%	3%	3%	726	U
M1	0+131 Mμ	483 K	84 K	16 K	12 K	23.2 → 13.4	33%	1%	0%	1	U
Q5	0+143 Mμ	708 K	157 K	13 K	11 K	28.8 → 23.6	21%	2%	2%	6	S
L1	0+157 Mμ	597 K	139 K	21 K	18 K	36.7 → 19.0	60%	3%	0%	30	U
S2	0+176 Mμ	722 K	161 K	29 K	17 K	37.5 → 27.5	33%	3%	1%	8	U
S3	1+201 Mμ	2624 K	471 K	12 K	6 K	14.5 → 9.8	21%	1%	2%	1	U
Q4	0+213 Mμ	781 K	175 K	19 K	16 K	29.2 → 23.3	25%	3%	1%	6	S
L2	0+216 Mμ	588 K	136 K	23 K	20 K	36.2 → 17.4	75%	1%	0%	6	U
X3	0+235 Mμ	1000 K	191 K	61 K	25 K	37.7 → 19.3	34%	1%	2%	14	U
G3	0+251 Mμ	1035 K	145 K	12 K	9 K	57.9 → 28.1	42%	1%	0%	0	S
Q0	0+401 Mμ	1493 K	342 K	37 K	28 K	63.3 → 40.0	50%	0%	0%	14	S
Q1	0+464 Mμ	1516 K	343 K	41 K	33 K	63.0 → 41.0	45%	0%	0%	14	S
T4	0+546 Mμ	2716 K	544 K	202 K	18 K	218.3 → 61.5	83%	1%	0%	3018	S
Q9	0+555 Mμ	1409 K	343 K	152 K	71 K	26.7 → 20.6	3%	5%	2%	99	S
Q3	0+613 Mμ	1883 K	448 K	27 K	22 K	60.1 → 40.3	41%	1%	1%	7	S
W1	0+626 Mμ	848 K	208 K	71 K	63 K	20.8 → 13.4	5%	14%	1%	28	U
Q6	0+646 Mμ	1211 K	266 K	40 K	35 K	30.4 → 23.2	30%	1%	1%	2	S
M6	0+660 Mμ	1378 K	266 K	80 K	52 K	34.0 → 22.2	33%	1%	1%	59	S
B2	0+668 Mμ	906 K	216 K	96 K	75 K	17.1 → 13.2	4%	5%	2%	16	U
T6	1+668 Mμ	2355 K	291 K	34 K	25 K	41.4 → 19.1	57%	0%	1%	11	U
D4	0+669 Mμ	1009 K	186 K	35 K	28 K	55.7 → 15.9	70%	0%	0%	2	S
M5	0+677 Mμ	1183 K	219 K	73 K	48 K	32.6 → 20.2	37%	1%	1%	139	U
R1	0+756 Mμ	913 K	220 K	87 K	74 K	17.3 → 12.4	3%	8%	0%	9	S
F1	0+859 Mμ	1485 K	311 K	218 K	135 K	17.6 → 15.1	1%	3%	3%	6	U
O2	7+1069 Mμ	18951 K	3144 K	3 K	2 K	17.0 → 9.5	35%	0%	0%	1	S
Q8	0+1107 Mμ	1786 K	437 K	184 K	109 K	29.4 → 20.2	6%	6%	1%	109	S

name	runtime	bytes	cells	nodes	learned	of size	triv	disc	sub	flushes	sat?
C5	0+1127 Mμ	1987 K	419 K	159 K	104 K	24.4 → 16.5	12%	2%	1%	776	S
D2	0+1159 Mμ	962 K	177 K	54 K	45 K	51.8 → 11.5	73%	0%	0%	2	U
C3	0+1578 Mμ	2375 K	571 K	190 K	96 K	49.7 → 23.4	39%	3%	2%	11	S
D1	0+1707 Mμ	1172 K	230 K	76 K	62 K	45.1 → 11.6	73%	0%	0%	2	U
T5	1+1735 Mμ	3658 K	617 K	80 K	59 K	72.5 → 40.9	50%	0%	0%	43	S
Q7	0+1761 Mμ	2055 K	419 K	515 K	118 K	33.9 → 20.3	9%	7%	0%	12	S
D3	0+1807 Mμ	1283 K	254 K	77 K	64 K	57.3 → 14.0	80%	0%	0%	1	S
R2	0+1886 Mμ	1220 K	296 K	173 K	149 K	17.0 → 11.8	3%	9%	0%	14	U
O1	7+2212 Mμ	18928 K	3140 K	5 K	3 K	17.3 → 8.9	39%	0%	0%	4	U
W3	0+2422 Mμ	1819 K	448 K	191 K	174 K	19.3 → 15.5	2%	12%	1%	18	S
P2	0+2435 Mμ	2039 K	504 K	378 K	301 K	20.9 → 13.7	3%	11%	1%	45	U
C6	0+2792 Mμ	2551 K	560 K	305 K	217 K	27.0 → 17.0	20%	2%	1%	492	U
E1	0+2902 Mμ	2116 K	453 K	180 K	144 K	38.0 → 20.5	21%	18%	0%	2	S
P1	0+3280 Mμ	2726 K	674 K	819 K	549 K	18.2 → 14.4	0%	9%	3%	45	U
G6	1+3941 Mμ	3523 K	647 K	380 K	253 K	31.0 → 17.8	31%	0%	0%	0	S
C9	13+4220 Mμ	35486 K	4923 K	116 K	32 K	11.8 → 9.9	5%	1%	1%	4986	S
C2	0+4625 Mμ	2942 K	712 K	442 K	255 K	46.1 → 18.8	42%	4%	1%	15	U
K4	0+5122 Mμ	1858 K	446 K	267 K	241 K	19.6 → 13.7	19%	2%	1%	5	U
C1	0+5178 Mμ	2532 K	613 K	510 K	311 K	48.9 → 17.0	48%	6%	1%	20	U
G7	1+6070 Mμ	4227 K	771 K	546 K	369 K	32.5 → 17.6	35%	0%	0%	0	U
C8	13+6081 Mμ	35014 K	4823 K	151 K	58 K	15.3 → 10.7	15%	1%	1%	8067	U
T7	1+6467 Mμ	5428 K	544 K	333 K	108 K	26.8 → 15.3	32%	1%	1%	14565	S
C7	8+7029 Mμ	20971 K	3174 K	908 K	32 K	9.5 → 8.4	0%	3%	0%	4965	S
T8	1+7046 Mμ	5322 K	517 K	356 K	117 K	26.9 → 15.0	33%	0%	1%	15026	U
W2	0+7785 Mμ	3561 K	884 K	501 K	432 K	34.7 → 21.3	13%	17%	1%	28	S
G5	1+7799 Mμ	4312 K	844 K	642 K	446 K	33.4 → 17.4	39%	0%	0%	0	U
G1	0+8681 Mμ	5052 K	1221 K	631 K	350 K	61.1 → 34.1	38%	1%	2%	55	S
K1	0+9813 Mμ	2864 K	685 K	405 K	360 K	36.2 → 18.4	53%	2%	0%	13	U
X7	1+11857 Mμ	6235 K	697 K	1955 K	224 K	40.6 → 23.7	35%	0%	1%	31174	S
K0	0+11997 Mμ	3034 K	731 K	493 K	421 K	35.6 → 19.4	45%	2%	0%	14	U
K2	0+12601 Mμ	3028 K	729 K	500 K	427 K	34.8 → 18.0	46%	2%	0%	12	U
A2	0+13947 Mμ	3766 K	843 K	645 K	585 K	34.4 → 15.9	32%	1%	0%	0	S
K8	0+15033 Mμ	2748 K	680 K	821 K	699 K	21.2 → 13.1	8%	15%	1%	93	U
P4	0+16907 Mμ	6936 K	1721 K	1676 K	1314 K	36.5 → 24.0	5%	11%	1%	33	S
A1	0+17073 Mμ	3647 K	815 K	763 K	701 K	30.7 → 14.7	29%	2%	0%	0	U
T3	0+19266 Mμ	10034 K	2373 K	2663 K	323 K	291.8 → 72.9	86%	1%	0%	34265	U
K7	0+20577 Mμ	3168 K	721 K	1286 K	828 K	23.3 → 13.5	9%	15%	0%	9	U
K3	0+20990 Mμ	3593 K	878 K	453 K	407 K	36.7 → 19.0	55%	2%	0%	6	U
W4	0+21295 Mμ	3362 K	834 K	977 K	899 K	19.0 → 14.1	4%	15%	0%	21	U
M8	0+22281 Mμ	4105 K	994 K	992 K	785 K	37.3 → 20.5	43%	1%	1%	6	U
G2	0+23424 Mμ	6910 K	1685 K	1198 K	701 K	68.8 → 34.3	47%	1%	1%	120	U
D5	0+24141 Mμ	3232 K	779 K	787 K	654 K	63.5 → 13.4	78%	0%	0%	2	U
M7	0+24435 Mμ	4438 K	1077 K	1047 K	819 K	40.6 → 23.3	42%	1%	1%	6	S
C4	1+31898 Mμ	8541 K	2108 K	1883 K	1148 K	60.6 → 25.7	42%	4%	1%	12	S
G8	7+35174 Mμ	24854 K	2992 K	4350 K	1101 K	48.0 → 34.7	9%	0%	0%	1523	S
E2	0+53739 Mμ	5454 K	1258 K	2020 K	1658 K	41.5 → 20.8	25%	21%	0%	3	S
X8	2+248789 Mμ	12814 K	2311 K	17005 K	3145 K	56.4 → 22.5	63%	0%	0%	330557	U
L3	0+295571 Mμ	19653 K	4894 K	7402 K	6886 K	70.7 → 31.0	63%	8%	0%	30	U
L4	0+677815 Mμ	22733 K	5664 K	8545 K	7931 K	78.6 → 35.4	86%	0%	0%	5	U

Table 7 summarizes the salient statistics, again listing all cases in order of their median running time (exclusive of input and output). Each running time is actually broken into two parts, '$x+y$', where x is the time to initialize the data structures in step C1 and y is the time for the other steps, both rounded to megamems. For example, the exact median processing time for case L5 was 1,484,489μ to initialize, then 655,728μ to find a solution; this is shown as '1+1 Mμ' in the third line of the table. The time for initialization is usually negligible except when there are many clauses, as in problem N1.

The median run of problem L5 also allocated 1,841,372 bytes of memory for data; this total includes the space needed for 210,361 cells in the MEM array, at 4 bytes per cell, together with other arrays such as VAL, OVAL, HEAP, etc. The implementation considered here keeps unlearned binary clauses in a separate BIMP table, as explained in the answer to exercise 267.

This run of L5 found a solution after implicitly traversing a search tree with 138 "nodes." The number of nodes, or "decisions," is the number of times step C6 of the algorithm goes to step C3. It is shown as '0 K' in Table 7, because the node counts, byte counts, and cell counts are rounded to the nearest thousand.

The number of nodes always exceeds or equals the number of learned clauses, which is the number of conflicts detected at levels $d > 0$. (See step C7.) In the case of problem L5, only 84 clauses were learned; so again the table reports '0 K'. These 84 clauses had average length $r+1 = 146.1$; then the simplification process of exercise 257 reduced this average to just 38.4. Nevertheless, the resulting simplified clauses were still sufficiently long that the "trivial" clauses discussed in exercise 269 were sometimes used instead; this substitution happened 43 times (51 %). Furthermore 19 of the learned clauses (23 %) were immediately discarded, using the method of exercise 271. These percentages show up in the 'triv' and 'disc' columns of the table.

Sometimes, as in problems D1–D5, a large majority of the learned clauses were replaced by trivial ones; on the other hand, 27 of the 100 cases turned out to be less than 10 % trivial in this sense. Table 7 also shows that the discard rate was 5 % or more in 26 cases. The 'sub' column refers to learned clauses that were "subsumed on the fly" by the technique of exercise 270; this optimization is less common, yet it occurs often enough to be worthwhile.

The great variety in our examples is reflected in the variety of behaviors exhibited in Table 7, although several interesting trends can also be perceived. For example, the number of nodes is naturally correlated with the number of learned clauses, and both statistics tend to grow as the total running time increases. But there are significant exceptions: Two outliers, O1 and O2, have a remarkably high ratio of mems per learned clause, because of their voluminous data.

The penultimate column of Table 7 counts how often Algorithm C decided to restart itself after flushing unproductive literals from its current trail. This quantity does not simply represent the number of times step C5 discovers that $M \geq M_{\rm f}$; it depends also on the current agility level (see (127)) and on the parameter ψ in Table 4. Some problems, like A1 and A2, had such high agility that they were solved satisfactorily with no restarts whatsoever; but another one, T4, finished in about 500 megamems after restarting more than 3000 times.

The number of "purges" (recycling phases) is not shown, but it can be estimated from the number of learned clauses (see exercise 508). An aggressive purging policy has kept the total number of memory cells comfortably small.

Tuning up the parameters. Table 7 shows that the hardest problem of all for Algorithm C in these experiments, L4, found itself substituting trivial clauses 86 % of the time but making only 5 restarts. That test case would probably have

MEM
binary clauses
BIMP
search tree
decision tree, see search tree
nodes
decisions
learned clauses
conflicts
levels
trivial
subsumed on the fly
on-the-fly subsumption
restart
flushing
trail
agility level
purges
recycling phases
tuning of parameters–
parameters, tuning of–

been solved much more quickly if the algorithm's parameters had been specially adjusted for instances of the Langford problem.

Algorithm C, as implemented in the experiments above, has ten major parameters that can be modified by the user on each run:

- α, tradeoff between p and q in clause `RANGE` scores (see Eq. (123));
- ρ, damping factor in variable `ACT` scores (see after (118));
- ϱ, damping factor in clause `ACT` scores (see Eq. (125));
- $\Delta_{\rm p}$, initial value of the purging threshold $M_{\rm p}$ (see after (125));
- $\delta_{\rm p}$, amount of gradual increase in $M_{\rm p}$ (see after (125));
- τ, threshold used to prefer trivial clauses (see answer to exercise 269);
- w, full "warmup" runs done after a restart (see answer to exercise 287);
- p, probability of choosing a decision variable at random (see exercise 266);
- P, probability that `OVAL`(k) is initially even;
- ψ, agility threshold for flushing (see Table 4).

The values for these parameters initially came from seat-of-the-pants guesses

$$\alpha=0.2,\ \ \rho=0.95,\ \ \varrho=0.999,\ \ \Delta_{\rm p}=20000,\ \ \delta_{\rm p}=500,\qquad \tau=1,\ \ w=0,\ \ p=0.02,\ \ P=0,\ \ \psi=0.166667; \tag{193}$$

and these defaults gave reasonably good results, so they were used happily for many months (although there was no good reason to believe that they couldn't be improved). Then finally, after the author had assembled the set of 100 test cases in Table 6, it was time to decide whether to recommend the default values (193) or to come up with a better set of numbers.

Parameter optimization for general broad-spectrum use is a daunting task, not only because of significant differences between species of SAT instances but also because of the variability due to random choices when solving any specific instance. It's hard to know whether a change of parameter will be beneficial or harmful, when running times are so highly erratic. Ouch — Fig. 54 illustrates dramatic variations even when all ten parameters are held fixed, and only the seed for random numbers is changed! Furthermore the ten parameters are not at all independent: An increase in ρ, say, might be a good thing, but only if the other nine parameters are also modified appropriately. How then could *any* set of defaults be recommended, without an enormous expense of time and money?

Fortunately there's a way out of this dilemma, thanks to advances in the theory of learning. F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle have developed a tool called ParamILS intended specifically for making such tuneups [*J. Artificial Intelligence Research* **36** (2009), 267–306]; the 'ILS' in this name stands for "iterated local search." The basic idea is to start with a representative training set of not-too-hard problems, and to carry out random walks in the 10-dimensional parameter space using sophisticated refinements of WalkSAT-like principles. The best parameters discovered during this training session are then evaluated on more difficult problems outside the training set.

Langford problem
`RANGE` scores
tradeoffs
damping factors
`ACT` scores
activity scores
purging threshold
trivial clauses
warmup runs
restarts
random decision variables
initial guess for literals
`OVAL`
agility threshold
defaults
author
Hutter
Hoos
Leyton-Brown
Stützle
ParamILS
ILS
iterated local search
training set
random walks
WalkSAT

In March 2015, Holger Hoos helped the author to tune Algorithm C using ParamILS. The resulting parameters then yielded Fig. 54, and Table 7, and many other runtime values discussed above and below. Our training set consisted of 17 problems that usually cost less than 200 Mμ with the original parameters (193), namely $\{K5, K6, M2, M4, N1, S1, S4, X4, X6\}$ together with stripped-down versions of $\{A1, C2, C3, D1, D2, D3, D4, K0\}$. For example, instead of the vector $x_1 \ldots x_{100}$ required by problem A1, we looked only for a shorter vector $x = x_1 \ldots x_{62}$, now with $\nu x = 20$; instead of D1 and D2 we sought 13-bit factors of 31415926; instead of K0 we tried to 9-color the SGB graph `jean`.

Ten independent training runs with ParamILS gave ten potential parameter settings $(\alpha_i, \rho_i, \ldots, \psi_i)$. We evaluated them on our original 17 benchmarks, together with 25 others that were a bit more difficult: $\{F1, F2, S2, S3, T4, X5\}$, plus less-stripped-down variants of $\{A1, A2, A2, C7, C7, D3, D4, F1, F2, G1, G1, G2, G2, G8, K0, O1, O2, Q0, Q2\}$. For each of the ten shortlisted parameter settings, we ran each of these $17 + 25$ problems with each of the random seeds $\{1, 2, \ldots, 25\}$. Finally, hurray, we had a winner: The parameters $(\alpha_i, \rho_i, \ldots, \psi_i)$ with minimum total running time in this experiment were

$$\alpha = 0.4,\ \rho = 0.9,\ \varrho = 0.9995,\ \Delta_{\rm p} = 1000,\ \delta_{\rm p} = 500,\\ \tau = 10,\ w = 0,\ p = 0.02,\ P = 0.5,\ \psi = 0.05. \qquad (194)$$

And these are now the recommended defaults for general-purpose use.

How much have we thereby gained? Figure 55 compares the running times of our 100 examples, before and after tuning. It shows that the vast majority — 77 of them — now run faster; these are the cases to the right of the dotted line from $(1\,{\rm M}\mu, 1\,{\rm M}\mu)$ to $(1\,{\rm T}\mu, 1\,{\rm T}\mu)$. Half of the cases experience a speedup exceeding 1.455; 27 of them now run more than twice as fast as they previously did.

Of course every rule has exceptions. The behavior of case P4 has gotten spectacularly worse, almost three orders of magnitude slower! Indeed, we saw earlier in Fig. 54 that this case has an amazingly unstable running time; further peculiarities of P4 are discussed in exercise 511.

Our other major SAT solver, Algorithm L, also has parameters, notably

- α, magic tradeoff coefficient in heuristic scores (see Eq. (64));
- β, damping factor for double-look triggering (see step Y1);
- γ, clause weight per literal in heuristic scores (see exercise 175);
- ε, offset in heuristic scores (see answer to exercise 146);
- Θ, maximum heuristic score threshold (see answer to exercise 145);
- Y, maximum depth of double-lookahead (see step Y1).

ParamILS suggests the following default values, which have been used in Fig. 53:

$$\alpha = 3.5,\ \beta = 0.9998,\ \gamma = 0.2,\ \varepsilon = 0.001,\ \Theta = 20.0,\ Y = 1. \qquad (195)$$

Returning to Fig. 55, notice that the change from (193) to (194) has substantially hindered cases G3 and G4, which are examples of test pattern generation. Evidently such clauses have special characteristics that make them prefer special

Hoos
author
training set
SGB
book graphs
forty-two...
defaults
tradeoffs
damping factors
heuristic scores
adaptive control
trigger
double-looking ahead
ATPG
test pattern generation

Hoos
author
waerden

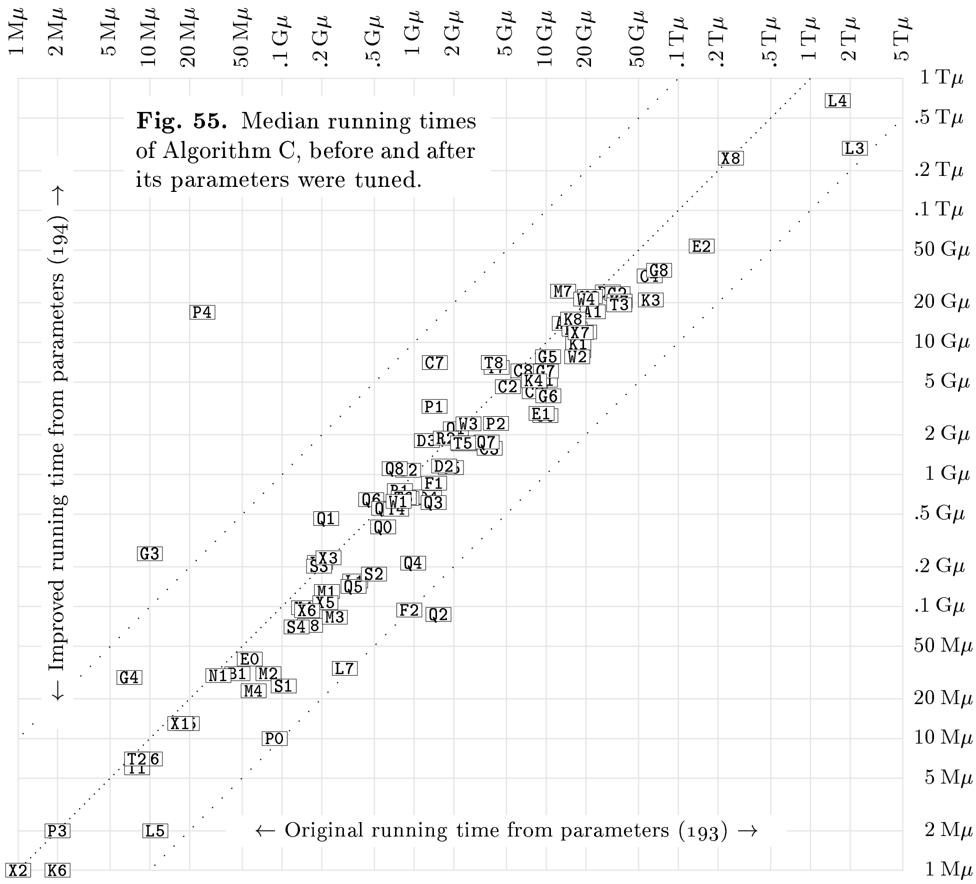


Fig. 55. Median running times of Algorithm C, before and after its parameters were tuned.

settings of the parameters. Our main reason for introducing parameters in the first place was, of course, to allow tweaking for different families of clauses.

Instead of finding values of $(\alpha, \rho, \ldots, \psi)$ that give good results in a broad spectrum of applications, we can clearly use a system like ParamILS to find values that are specifically tailored to a particular class of problems. In fact, this task is easier. For example, Hoos and the author asked for settings of the ten parameters that will tend to make Algorithm C do its best on problems of the form $waerden(3, k; n)$. A pair of ParamILS runs, based solely on the easy training cases $waerden(3, 9; 77)$ and $waerden(3, 10; 95)$, suggested the parameters

$$\alpha = 0.5,\ \ \rho = 0.9995,\ \ \varrho = 0.99,\ \ \Delta_\mathrm{p} = 100,\ \ \delta_\mathrm{p} = 10,$$
$$\tau = 10,\ \ w = 8,\ \ p = 0.01,\ \ P = 0.5,\ \ \psi = 0.15, \qquad (196)$$

and this set indeed works very well. Figure 56 shows typical details, with $7 \le k \le 14$ and with nine independent sample runs for every choice of k and n. Each unsatisfiable instance has $n = W(3, k)$, as given in the table following (10) above; each satisfiable instance has $n = W(3, k) - 1$. The fastest run using default

parallel methods
backtracking

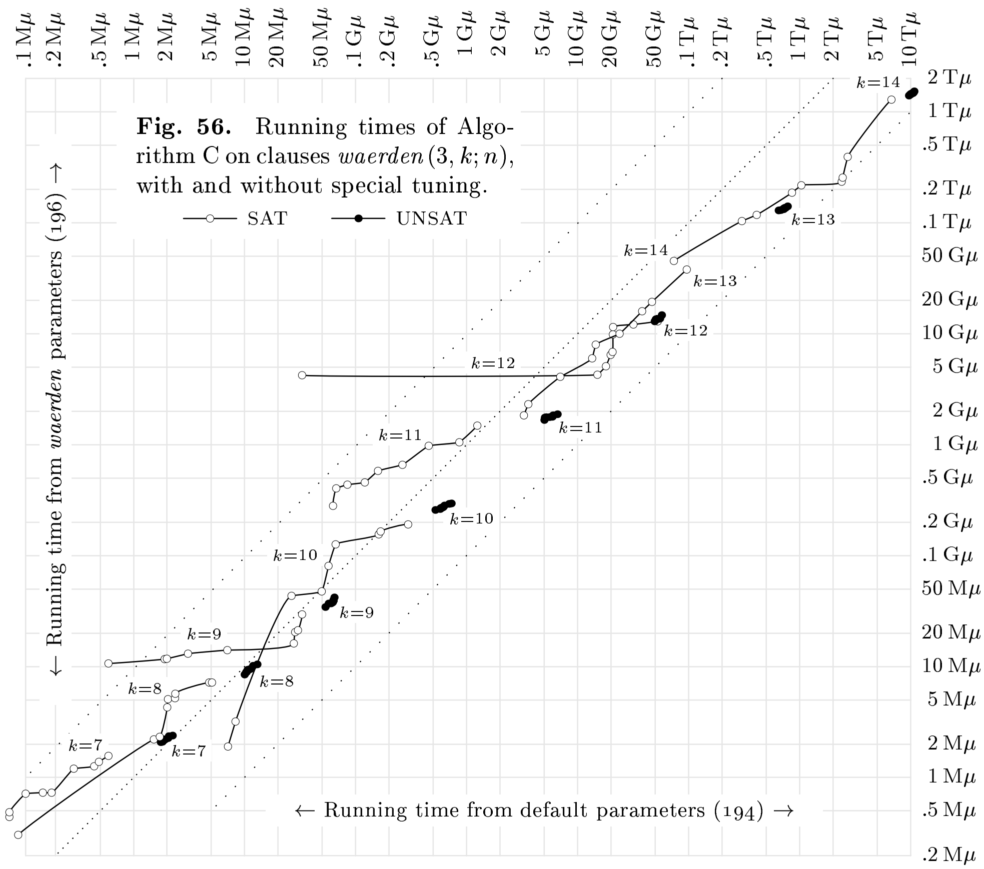


Fig. 56. Running times of Algorithm C on clauses *waerden*$(3, k; n)$, with and without special tuning.

parameters (194) has been paired in Fig. 56 with the fastest run using *waerden*-tuned parameters (196); similarly, the second-fastest, ..., second-slowest, and slowest runs have also been paired. Notice that satisfiable instances tend to take an unpredictable amount of time, as in Fig. 54. In spite of the fact that the new parameters (196) were found by a careful study of just two simple instances, they clearly yield substantial savings when applied to much, much harder problems of a similar nature. (See exercise 512 for another instructive example.)

Exploiting parallelism. Our focus in the present book is almost entirely on sequential algorithms, but we should be aware that the really tough instances of SAT are best solved by parallel methods.

Problems that are amenable to backtracking can readily be decomposed into subproblems that partition the space of solutions. For example, if we have 16 processors available, we can start them off on independent SAT instances in which variables $x_1x_2x_3x_4$ have been forced to equal 0000, 0001, ..., 1111.

A naïve decomposition of that kind is rarely the best strategy, however. Perhaps only one of those sixteen cases is really challenging. Perhaps some of

the processors are slower than others. Perhaps several processors will learn new clauses that the other processors ought to know. Furthermore, the splitting into subproblems need not occur only at the root of the search tree. Careful load-balancing and sharing of information will do much better. These challenges were addressed by a pioneering system called PSATO [H. Zhang, M. P. Bonacina, and J. Hsiang, *Journal of Symbolic Computation* **21** (1996), 543–560].

A much simpler approach should also be mentioned: We can start up many different solvers, or many copies of the same solver, with different sources of random numbers. As soon as one has finished, we can then terminate the others.

The best parallelized SAT solvers currently available are based on the "cube and conquer" paradigm, which combines conflict-driven clause learning with lookahead techniques that choose branch variables for partitioning; see M. J. H. Heule, O. Kullmann, S. Wieringa, and A. Biere, *LNCS* **7261** (2012), 50–65. In particular, this approach is excellent for the *waerden* problems.

PSATO
Zhang
Bonacina
Hsiang
random numbers
cube and conquer
conflict-driven clause learning
lookahead
Heule
Kullmann
Wieringa
Biere
waerden
DODGSON
tautologous
truth table
QUINE
syllogism
Socrates
resolution
Boole
Dodgson
Carroll
eliminate variables
Dodgson
Bartley
Method of Trees
Carroll
backtracking

> *Today has proved to be an epoch in my Logical work.*
> *. . . I think of calling it the 'Genealogical Method.'*
>
> — CHARLES L. DODGSON, *Diary* (16 July 1894)

> *The method of showing a statement to be tautologous*
> *consists merely of constructing a table under it in the usual way*
> *and observing that the column under the main connective*
> *is composed entirely of 'T's.*
>
> — W. V. O. QUINE, *Mathematical Logic* (1940)

A brief history. The classic syllogism "All men are mortal; Socrates is a man; hence Socrates is mortal" shows that the notion of *resolution* is quite ancient:

$$\neg\text{Man} \lor \text{Mortal};\quad \neg\text{Socrates} \lor \text{Man};\quad \therefore\ \neg\text{Socrates} \lor \text{Mortal}.$$

Of course, algebraic demonstrations that $(\neg x \lor y)\land(\neg z \lor x)$ implies $(\neg z \lor y)$, when x, y, and z are arbitrary Boolean expressions, had to wait until Boole and his 19th-century followers brought mathematics to bear on the subject. The most notable contributor, resolutionwise, was perhaps C. L. Dodgson, who spent the last years of his life working out theories of inference by which complex chains of reasoning could be analyzed by hand. He published *Symbolic Logic*, Part I, in 1896, addressing it to children and to the young-in-heart by using his famous pen name Lewis Carroll. Section VII.II.§3 of that book explains and illustrates how to eliminate variables by resolution, which he called the Method of Underscoring.

When Dodgson died unexpectedly at the beginning of 1898, his nearly complete manuscript for *Symbolic Logic*, Part II, vanished until W. W. Bartley III was able to resurrect it in 1977. Part II was found to contain surprisingly novel ideas — especially its Method of Trees, which would have completely changed the history of mechanical theorem proving if it had come to light earlier. In this method, which Carroll documented at length in a remarkably clear and entertaining way, he constructed search trees essentially like Fig. 39, then converted them into proofs by resolution. Instead of backtracking as in Algorithm D,

which is a recursive depth-first method, he worked breadth-first: Starting at the root, he exploited unit clauses when possible, and branched on binary (or even ternary) clauses when necessary, successively filling out all unfinished branches level-by-level in hopes of being able to reuse computations.

Logicians of the 20th century took a different tack. They basically dealt with the satisfiability problem in its equivalent dual form as the tautology problem, namely to decide when a Boolean formula is always true. But they dismissed tautology-checking as a triviality, because it could always be solved in a finite number of steps by just looking at the truth table. Logicians were far more interested in problems that were provably *unsolvable* in finite time, such as the halting problem — the question of whether or not an algorithm terminates. Nobody was bothered by the fact that an n-variable function has a truth table of length 2^n, which exceeds the size of the universe even when n is rather small.

Practical computations with disjunctive normal forms were pioneered by Archie Blake in 1937, who introduced the "consensus" of two implicants, which is dual to the resolvent of two clauses. Blake's work was, however, soon forgotten; E. W. Samson, B. E. Mills, and (independently) W. V. O. Quine rediscovered the consensus operation in the 1950s, as discussed in exercise 7.1.1–31.

The next important step was taken by E. W. Samson and R. K. Mueller [Report AFCRC-TR-55-118 (Cambridge, Mass.: Air Force Cambridge Research Center, 1955), 16 pages], who presented an algorithm for the tautology problem that uses consensus to eliminate variables one by one. Their algorithm therefore was equivalent to SAT solving by successively eliminating variables via resolution. Samson and Mueller demonstrated their algorithm by applying it to the unsatisfiable clauses that we considered in (112) above.

Independently, Martin Davis and Hilary Putnam had begun to work on the satisfiability problem, motivated by the search for algorithms to deduce formulas in first order logic — unlike Samson, Mills, and Mueller, who were chiefly interested in synthesizing efficient circuits. Davis and Putnam wrote an unpublished 62-page report "Feasible computational methods in the propositional calculus" (Rensselaer Polytechnic Institute, October 1958) in which a variety of different approaches were considered, such as the removal of unit clauses and pure literals, as well as "case analysis," that is, backtracking with respect to the subproblems $F\,|\,x$ and $F\,|\,\bar{x}$. As an alternative to case analysis, they also discussed eliminating the variable x by resolution. The account of this work that was eventually published [*JACM* **7** (1960), 201–215] concentrated on hand calculation, and omitted case analysis in favor of resolution; but when the process was later implemented on a computer, jointly with George Logemann and Donald Loveland [*CACM* **5** (1962), 394–397], the method of backtracking through different cases was found to work better with respect to memory requirements. (See Davis's account of these developments in *Handbook of Automated Reasoning* (2001), 3–15.)

This early work didn't actually cause the satisfiability problem to appear on many people's mental radar screens, however. Far from it; ten years went by before SAT became an important buzzword. The picture changed in 1971, when Stephen A. Cook showed that satisfiability is the key to solving NP-

recursive
depth-first
breadth-first
dual form
tautology problem
unsolvable
halting problem
disjunctive normal forms
Blake
consensus
resolvent
Samson
Mills
Quine
Mueller
Davis
Putnam
first order logic
Samson
Mills
Mueller
unit clauses
pure literals
resolution
Logemann
Loveland
Davis
Cook
NP-complete problems

complete problems: He proved that any algorithm to solve a decision problem in nondeterministic polynomial time can be represented efficiently as a conjunction of ternary clauses to be satisfied. (See *STOC* **3** (1971), 151–158. We'll study NP-completeness in Section 7.9.) Thus, a great multitude of hugely important problems could all be solved rather quickly, if we could only devise a decent algorithm for a *single* problem, 3SAT; and 3SAT seemed almost absurdly simple to solve.

A year of heady optimism following the publication of Cook's paper soon gave way to the realization that, alas, 3SAT might not be so easy after all. Ideas that looked promising in small cases didn't scale well, as the problem size was increased. Hence the central focus of work on satisfiability largely retreated into theoretical realms, unrelated to programming practice, except for occasional studies that used SAT as a simple model for the behavior of backtracking algorithms in general. Examples of such investigations, pioneered by A. T. Goldberg, P. W. Purdom, Jr., C. A. Brown, J. V. Franco, and others, appear in exercises 213–216. See P. W. Purdom, Jr., and G. N. Haven, *SICOMP* **26** (1997), 456–483, for a survey of subsequent progress on questions of that kind.

The state of SAT art in the early 90s was well represented by an international programming competition held in 1992 [see M. Buro and H. Kleine Büning, *Bulletin EATCS* **49** (February 1993), 143–151]. The winning programs in that contest can be regarded as the first successful lookahead solvers on the path from Algorithm A to Algorithm L. Max Böhm "took the gold" by choosing the next branch variable based on lexicographically maximal $(H_1(x),\ldots,H_n(x))$, where

$$H_k(x) = h_k(x)+h_k(\bar x)+\min\bigl(h_k(x),h_k(\bar x)\bigr),\;\; h_k(x) = \bigl|\{C\in F \mid x\in C, |C|=k\}\bigr|.$$

[See M. Böhm and E. Speckenmeyer, *Ann. Math. Artif. Intelligence* **17** (1996), 381–400. A. Rauzy had independently proposed a somewhat similar branching criterion in 1988; see *Revue d'intelligence artificielle* **2** (1988), 41–60.] The silver medal went to Hermann Stamm, who used strong components of the dependency digraph to narrow the search at each branch node.

Advances in practical algorithms for satisfiability now began to take off. The benchmark programs of 1992 had been chosen at random, but the DIMACS Implementation Challenge of 1993 featured also a large number of structured instances of SAT. The main purpose of this "challenge" was not to crown a winner, but to bring more than 100 researchers together for a three-day workshop, at which they could compare and share results. In retrospect, the best overall performance at that time was arguably achieved by an elaborate lookahead solver called C-SAT, which introduced techniques for detailed exploration of the first-order effects of candidate literals [see O. Dubois, P. Andre, Y. Boufkhad, and J. Carlier, *DIMACS* **26** (1996), 415–436]. Further refinements leading towards the ideas in Algorithm L appeared in a Ph.D. thesis by Jon W. Freeman (Univ. of Pennsylvania, 1995), and in the work of Chu Min Li, who introduced double lookahead [see *Information Processing Letters* **71** (1999), 75–80]. The weighted binary heuristic (67) was proposed by O. Dubois and G. Dequen, *Proc. International Joint Conference on Artificial Intelligence* **17** (2001), 248–253.

nondeterministic polynomial time
3SAT
Goldberg
Purdom
Brown
Franco
Haven
competition
Buro
Kleine Büning
contest
lookahead solvers
Böhm
Speckenmeyer
Rauzy
Stamm
strong components
dependency digraph
C-SAT
candidate literals
Dubois
Andre
Boufkhad
Carlier
Freeman
Li
double lookahead
Dubois
Dequen

Meanwhile the ideas underlying Algorithm C began to emerge. João P. Marques-Silva, in his 1995 thesis directed by Karem A. Sakallah, discovered how to turn unit-propagation conflicts into one or more clauses learned at "unique implication points," after which it was often possible to backjump past decisions that didn't affect the conflict. [See *IEEE Trans.* **C48** (1999), 506–521.] Similar methods were developed independently by R. J. Bayardo, Jr., and R. C. Schrag [*AAAI Conf.* **14** (1997), 203–208], who considered only the special case of clauses that include the current decision literal, but introduced techniques for purging a learned clause when one of its literals was forced to flip its value. Both groups limited the size of learned clauses, and noticed that their new methods gave significant speedups on benchmark problems related to industrial applications.

The existence of fast SAT solvers, coupled with Gunnar Stålmarck's new ideas about applying logic to computer design [see Swedish patent 467076 (1992)], led to the introduction of bounded model checking techniques by Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu [*LNCS* **1579** (1999), 193–207]. Satisfiability techniques had also been introduced to solve classical planning problems in artificial intelligence [Henry Kautz and Bart Selman, *Proc. European Conf. Artificial Intelligence* **10** (1992), 359–363]. Designers could now verify much larger models than had been possible with BDD methods.

The major breakthroughs appeared in a solver called Chaff [M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, *ACM/IEEE Design Automation Conf.* **38** (2001), 530–535], which had two especially noteworthy innovations: (i) "VSIDS" (the Variable State Independent Decreasing Sum heuristic), a surprisingly effective way to select decision literals, which also worked well with restarts, and which suggested the even better `ACT` heuristic of Algorithm C that soon replaced it; also (ii) lazy data structures with two watched literals per clause, which made unit propagation much faster with respect to large learned clauses. (A somewhat similar watching scheme, introduced earlier by H. Zhang and M. Stickel [*J. Automated Reasoning* **24** (2000), 277–296], had the disadvantage that it needed to be downdated while backtracking.)

These exciting developments sparked a revival of international SAT competitions, which have been held annually since 2002. The winner in 2002, BerkMin by E. Goldberg and Y. Novikov, has been described well in *Discrete Applied Mathematics* **155** (2007), 1549–1561. And year after year, these challenging contests have continued to spawn further progress. By 2010, more than twice as many benchmarks could be solved in a given period of time as in 2002, using the programs of 2002 and 2010 on the computers of 2010 [see M. Järvisalo, D. Le Berre, O. Roussel, and L. Simon, *AI Magazine* **33**,1 (Spring 2012), 89–94].

The overall champion in 2007 was `SATzilla`, which was actually not a separate SAT solver but rather a program that knew how to choose intelligently between *other* solvers on any given instance. `SATzilla` would first take a few seconds to compute basic features of a problem: the distribution of literals per clause and clauses per literal, the balance between positive and negative occurrences of variables, the proximity to Horn clauses, etc. Samples could quickly be taken to estimate how many unit propagations occur at levels 1, 4, 16, 64, 256, and how

Marques-Silva
Sakallah
unit-propagation
conflicts
unique implication points
UIP
backjump
Bayardo
Schrag
decision literal
purging
learned clause
Stålmarck
bounded model checking
Biere
Cimatti
Clarke
Zhu
planning
Kautz
Selman
BDD
Chaff
Moskewicz
Madigan
Zhao
Zhang
Malik
VSIDS
restarts
flushing
`ACT`
activity scores
watched literals
unit propagation
Zhang
Stickel
backtracking
competitions
BerkMin
Goldberg
Novikov
Järvisalo
Le Berre
Roussel
Simon
`SATzilla`
Horn clauses
unit propagations

many decisions are needed before reaching a conflict. Based on these numbers, and experience with the performance of the other solvers on the previous year's benchmarks, SATzilla was trained to select the algorithm that appeared most likely to succeed. This "portfolio" approach, which tunes itself nicely to the characteristics of vastly different sets of clauses, has continued to dominate the international competitions ever since. Of course portfolio solvers rely on the existence of "real" solvers, invented independently and bug-free, which shine with respect to particular classes of problems. And of course the winner of competitions may not be the best actual system for practical use. [See L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown, *J. Artificial Intelligence Research* **32** (2008), 565–606; *LNCS* **7317** (2012), 228–241; *CACM* **57**, 5 (May 2014), 98–107.]

Historical notes about details of the algorithms, and about important related techniques such as preprocessing and encoding, have already been discussed above as the algorithms and techniques were described.

One recurring theme appears to be that the behavior of SAT solvers is full of surprises: Some of the most important improvements have been introduced for what has turned out to be the wrong reasons, and a theoretical understanding is still far from adequate.

[In future, the next breakthrough might come from "variable learning," as suggested by Tseytin's idea of extended resolution: Just as clause learning increases the number of clauses, m, we might find good ways to increase the number of variables, n. The subject seems to be far from fully explored.]

portfolio
tunes itself
Xu
Hutter
Hoos
Leyton-Brown
Tseytin
extended resolution
Pincusians
waerden
Local Lemma
$W(k_0, k_1, \ldots, k_{b-1})$
monotonic
binary

EXERCISES

1. [*10*] What are the shortest (a) satisfiable (b) unsatisfiable sets of clauses?

2. [*20*] Travelers to the remote planet Pincus have reported that all the healthy natives like to dance, unless they're lazy. The lazy nondancers are happy, and so are the healthy dancers. The happy nondancers are healthy; but natives who are lazy and healthy aren't happy. Although the unhappy, unhealthy ones are always lazy, the lazy dancers are healthy. What can we conclude about Pincusians, based on these reports?

3. [*M21*] Exactly how many clauses are in $waerden(j, k; n)$?

4. [*22*] Show that the 32 constraints of $waerden(3, 3; 9)$ in (9) remain unsatisfiable even if up to four of them are removed.

5. [*M46*] Is $W(3, k) = \Theta(k^2)$?

▶ **6.** [*HM37*] Use the Local Lemma to show that $W(3, k) = \Omega(k^2/(\log k)^3)$.

7. [*21*] Can one satisfy the clauses $\{(x_i \vee x_{i+2^d} \vee x_{i+2^{d+1}}) \mid 1 \le i \le n - 2^{d+1}, d \ge 0\} \cup \{(\bar{x}_i \vee \bar{x}_{i+2^d} \vee \bar{x}_{i+2^{d+1}}) \mid 1 \le i \le n - 2^{d+1}, d \ge 0\}$?

▶ **8.** [*20*] Define clauses $waerden(k_0, k_1, \ldots, k_{b-1}; n)$ that are satisfiable if and only if $n < W(k_0, k_1, \ldots, k_{b-1})$.

9. [*24*] Determine the value of $W(2, 2, k)$ for all $k \ge 0$. *Hint:* Consider $k \bmod 6$.

▶ **10.** [*21*] Show that every satisfiability problem with m clauses and n variables can be transformed into an equivalent monotonic problem with $m+n$ clauses and $2n$ variables, in which the first m clauses have only negative literals, and the last n clauses are binary with two positive literals.

11. [*27*] (M. Tsimelzon, 1994.) Show that a general 3SAT problem with clauses $\{C_1,\ldots,C_m\}$ and variables $\{1,\ldots,n\}$ can be reduced to a 3D MATCHING problem of size $10m$ that involves the following cleverly designed triples:

Each clause C_j corresponds to 3×10 vertices, namely lj, $\bar{l}j$, $|l|j'$, and $|l|j''$ for each $l\in C_j$, together with wj, xj, yj, and zj, and also $j'k$ and $j''k$ for $1\le k\le 7$. If i or $\bar{\imath}$ occurs in t clauses $C_{j_1},\ldots,C_{j_t}$, there are t "true" triples $\{ij_k, ij'_k, ij''_k\}$ and t "false" triples $\{\bar{\imath}j_k, ij'_k, ij''_{1+(k \bmod t)}\}$, for $1\le k\le t$. Each clause $C_j=(l_1\vee l_2\vee l_3)$ also spawns three "satisfiability" triples $\{\bar{l}_1j, j'1, j''1\}$, $\{\bar{l}_2j, j'1, j''2\}$, $\{\bar{l}_3j, j'1, j''3\}$; six "filler" triples $\{l_1j, j'2, j''1\}$, $\{\bar{l}_1j, j'3, j''1\}$, $\{l_2j, j'4, j''2\}$, $\{\bar{l}_2j, j'5, j''2\}$, $\{l_3j, j'6, j''3\}$, $\{\bar{l}_3j, j'7, j''3\}$; and twelve "gadget" triples $\{wj, j'2, j''4\}$, $\{wj, j'4, j''4\}$, $\{wj, j'6, j''4\}$, $\{xj, j'2, j''5\}$, $\{xj, j'5, j''5\}$, $\{xj, j'7, j''5\}$, $\{yj, j'3, j''6\}$, $\{yj, j'4, j''6\}$, $\{yj, j'7, j''6\}$, $\{zj, j'3, j''7\}$, $\{zj, j'5, j''7\}$, $\{zj, j'6, j''7\}$. Thus there are $27m$ triples altogether.

For example, Rivest's satisfiability problem (6) leads to a 3D matching problem with 216 triples on 240 vertices; the triples that involve vertices 18 and $\bar{1}8$ are $\{18, 18', 18''\}$, $\{\bar{1}8, 18', 11''\}$, $\{\bar{1}8, 8'1, 8''2\}$, $\{18, 8'4, 8''2\}$, $\{\bar{1}8, 8'5, 8''2\}$.

12. [*21*] (M. J. H. Heule.) Simplify (13) by exploiting the identity

$$S_{\le 1}(y_1,\ldots,y_p) \;=\; \exists t\,\bigl(S_{\le 1}(y_1,\ldots,y_j,t)\wedge S_{\le 1}(\bar{t},y_{j+1},\ldots,y_p)\bigr).$$

13. [*24*] Exercise 7.2.2.1–00 defines an exact cover problem that corresponds to Langford pairs of order n. (See page vii.)

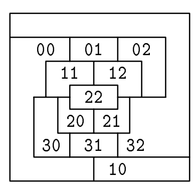
a) What are the constraints analogous to (12) when $n=4$?

b) Show that there's a simple way to avoid duplicate binary clauses such as those in (14), whenever an exact cover problem is converted to clauses using (13).

c) Describe the corresponding clauses *langford*(4) and *langford*′(4).

14. [*22*] Explain why the clauses (17) might help a SAT solver to color a graph.

15. [*24*] By comparing the McGregor graph of order 10 in Fig. 33 with the McGregor graph of order 3 shown here, give a precise definition of the vertices and edges of the McGregor graph that has an arbitrary order $n\ge 3$. Exactly how many vertices and edges are present in this graph, as a function of n?



16. [*21*] Do McGregor graphs have cliques of size 4?

17. [*26*] Let $f(n)$ and $g(n)$ be the smallest and largest values of r such that McGregor's graph of order n can be 4-colored, and such that some color appears exactly r times. Use a SAT solver to find as many values of $f(n)$ and $g(n)$ as you can.

▸ **18.** [*28*] By examining the colorings found in exercise 17, define an explicit way to 4-color a McGregor graph of *arbitrary* order n, in such a way that one of the colors is used at most $\frac{5}{6}n$ times. *Hint:* The construction depends on the value of $n \bmod 6$.

▸ **19.** [*29*] Continuing exercise 17, let $h(n)$ be the largest number of regions that can be given two colors simultaneously (without using the clauses (17)). Investigate $h(n)$.

20. [*40*] In exactly how many ways can McGregor's map (Fig. 33) be four-colored?

21. [*22*] Use a SAT solver to find a minimum-size *kernel* in the graph of Fig. 33.

22. [*20*] Color the graph $\overline{C_5\boxtimes C_5}$ with the fewest colors. (Two vertices of this graph can receive the same color if and only if they are a king move apart in a 5×5 torus.)

23. [*20*] Compare the clauses (18) and (19) to (20) and (21) in the case $n=7$, $r=4$.

Tsimelzon
3D MATCHING problem
gadget
Rivest
Heule
symmetric threshold functions
encoding
exclusion clauses
at-most-one
exact cover problem
Langford pairs
langford(n) and *langford*′(n)
exclusion clauses
at-most-one
McGregor graph
kernel
strong product
complement of a graph
king move
torus

▸ **24.** [*M32*] The clauses obtained from (20) and (21) in the previous exercise can be simplified, because we can remove the two that contain the pure literal b_1^2.

a) Prove that the literal b_1^2 is *always* pure in (20) and (21), when $r > n/2$.

b) Show that b_1^2 might also be pure in some cases when $r < n/2$.

c) The clauses obtained from (20) and (21) have *many* pure literals b_j^k when r has its maximum value $n-1$. Furthermore, their removal makes other literals pure. How many clauses will remain in this case after all pure literals have been eliminated?

d) Show that the complete binary tree with $n \geq 2$ leaves is obtained from complete binary trees with n' and $n'' = n - n'$ leaves, where either n' or n'' is a power of 2.

e) Let $a(n,r)$ and $c(n,r)$ be respectively the number of auxiliary variables b_j^k and the total number of clauses that remain after all of the pure auxiliary literals have been removed from (20) and (21). What are $a(2^k, 2^{k-1})$ and $c(2^k, 2^{k-1})$?

f) Prove that $a(n,r) = a(n,n'') = a(n,n')$ for $n'' \leq r \leq n'$, and this common value is $\max_{1\leq r<n} a(n,r)$. Also $a(n,r) = a(n,n-r)$; and $c(n,r) \geq c(n,n-r)$ if $r \leq n/2$.

25. [*21*] Show that (18)–(19) and (20)–(21) are equally effective when $r = 2$.

26. [*22*] Prove that Sinz's clauses (18) and (19) enforce the cardinality constraint $x_1 + \cdots + x_n \leq r$. *Hint:* Show that they imply $s_j^k = 1$ whenever $x_1 + \cdots + x_{j+k-1} \geq k$.

27. [*20*] Similarly, prove the correctness of Bailleux and Boufkhad's (20) and (21). *Hint:* They imply $b_j^k = 1$ whenever the leaves below node k contain j or more 1s.

▸ **28.** [*20*] What clauses result from (18) and (19) when we want to ensure that $x_1 + \cdots + x_n \geq 1$? (This special case converts arbitrary clauses into 3SAT clauses.)

▸ **29.** [*20*] Instead of the single constraint $x_1 + \cdots + x_n \leq r$, suppose we wish to impose a sequence of constraints $x_1 + \cdots + x_i \leq r_i$ for $1 \leq i \leq n$. Can this be done nicely with additional clauses and auxiliary variables?

▸ **30.** [*22*] If auxiliary variables s_j^k are used as in (18) and (19) to make $x_1 + \cdots + x_n \leq r$, while s'^k_j are used to make $\bar{x}_1 + \cdots + \bar{x}_n \leq n - r$, show that we may unify them by taking $s'^j_k = \overline{s_j^k}$, for $1 \leq j \leq n-r$, $1 \leq k \leq r$. Can (20) and (21) be similarly unified?

▸ **31.** [*28*] Let $F_t(r)$ be the smallest n for which there is a bit vector $x_1 \ldots x_n$ with $x_1 + \cdots + x_n = r$ and with no t equally spaced 1s. For example, $F_3(12) = 30$ because of the unique solution 101100011010000000010110001101. Discuss how $F_t(n)$ might be computed efficiently with the help of a SAT solver.

32. [*15*] A *list coloring* is a graph coloring in which v's color belongs to a given set $L(v)$, for each vertex v. Represent list coloring as a SAT problem.

33. [*21*] A *double coloring* of a graph is an assignment of two distinct colors to every vertex in such a way that neighboring vertices share no common colors. Similarly, a q-tuple coloring assigns q distinct colors to each vertex. Find double and triple colorings of the cycle graphs C_5, C_7, C_9, ..., using as few colors as possible.

34. [*HM26*] The *fractional coloring number* $\chi^*(G)$ of a graph G is defined to be the minimum ratio p/q for which G has a q-tuple coloring that uses p colors.

a) Prove that $\chi^*(G) \leq \chi(G)$, and show that equality holds in McGregor's graphs.

b) Let $S_1, \ldots, S_N$ be all the independent subsets of G's vertices. Show that

$$\chi^*(G) = \min_{\lambda_1,\ldots,\lambda_N \geq 0} \bigl\{\lambda_1 + \cdots + \lambda_N \bigm| \textstyle\sum_{j=1}^N \lambda_j [v \in S_j] = 1 \text{ for all vertices } v\bigr\}.$$

(This is a fractional exact cover problem.)

c) What is the fractional coloring number $\chi^*(C_n)$ of the cycle graph C_n?

pure literal
complete binary tree
Sinz
cardinality constraint
Bailleux
Boufkhad
3SAT
auxiliary variables
arithmetic progressions, avoiding
list coloring
double coloring
coloring, multiple
cycle graphs
fractional coloring number
coloring, fractional
McGregor's graphs
fractional exact cover
exact cover, fractional

d) Consider the following greedy algorithm for coloring G: Set $k \leftarrow 0$ and $G_0 \leftarrow G$; while G_k is nonempty, set $k \leftarrow k+1$ and $G_k \leftarrow G_{k-1} \setminus C_k$, where C_k is a maximum independent set of G_{k-1}. Prove that $k \le H_{\alpha(G)}\chi^*(G)$, where $\alpha(G)$ is the size of G's largest independent set; hence $\chi(G)/\chi^*(G) \le H_{\alpha(G)} = O(\log n)$. *Hint:* Let $t_v = 1/|C_i|$ if $v \in C_i$, and show that $\sum_{v\in S} t_v \le H_{|S|}$ whenever S is an independent set.

35. [*22*] Determine $\chi^*(G)$ when G is (a) the graph of the contiguous United States (see 7–(17) and exercise 7–45); (b) the graph of exercise 22.

▸ **36.** [*22*] A *radio coloring* of a graph, also known as an $L(2,1)$ labeling, is an assignment of integer colors to vertices so that the colors of u and v differ by at least 2 when $u \mathrel{—} v$, and by at least 1 when u and v have a common neighbor. (This notion, introduced by Fred Roberts in 1988, was motivated by the problem of assigning channels to radio transmitters, without interference from "close" transmitters and without strong interference from "very close" transmitters.) Find a radio coloring of Fig. 33 that uses only 16 consecutive colors.

37. [*20*] Find an optimum radio coloring of the contiguous USA graph (see 7–(17)).

38. [*M25*] How many consecutive colors are needed for a radio coloring of (a) the $n \times n$ square grid $P_n \,\square\, P_n$? (b) the vertices $\{(x,y,z) \mid x,y,z \ge 0,\ x+y+z=n\}$, which form a triangular grid with $n+1$ vertices on each side.

39. [*M46*] Find an optimum radio coloring of the n-cube, for some value of $n > 6$.

40. [*01*] Is the factorization problem (22) unsatisfiable whenever z is a prime number?

41. [*M21*] Determine the number of Boolean operations $\wedge$, $\vee$, $\oplus$ needed to multiply m-bit numbers by n-bit numbers with Dadda's scheme, when $2 \le m \le n$.

42. [*21*] Tseytin encoding analogous to (24) can be devised also for ternary operations, without introducing any additional variables besides those of the function being encoded. Illustrate this principle by encoding the basic operations $x \leftarrow t \oplus u \oplus v$ and $y \leftarrow \langle tuv \rangle$ of a full adder directly, instead of composing them from $\oplus$, $\wedge$, and $\vee$.

▸ **43.** [*21*] For which integers $n \ge 2$ do there exist odd palindromic binary numbers $x = (x_n \ldots x_1)_2 = (x_1 \ldots x_n)_2$ and $y = (y_n \ldots y_1)_2 = (y_1 \ldots y_n)_2$ such that their product $xy = (z_{m+n} \ldots z_1)_2 = (z_1 \ldots z_{m+n})_2$ is also palindromic?

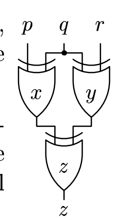
▸ **44.** [*30*] (Maximum ones.) Find the largest possible value of $\nu x + \nu y + \nu(xy)$, namely the greatest total number of 1 bits, over all multiplications of 32-bit binary x and y.

45. [*20*] Specify clauses that constrain $(z_t \ldots z_1)_2$ to be a perfect square.

46. [*30*] Find the largest perfect square less than 2^{100} that is a binary palindrome.

▸ **47.** [*20*] Suppose a circuit such as Fig. 34 has m outputs and n inputs, with g gates that transform two signals into one and h gates that transform one signal into two. Find a relation between g and h, by expressing the total number of wires in two ways.

48. [*20*] The small circuit shown here has three inputs, three XOR gates, one fanout gate, eight wires, and one output. Which single-stuck-at faults are detected by each of the eight test patterns pqr?



49. [*24*] Write a program that determines exactly which of the 100 single-stuck-at faults of the circuit in Fig. 34 are detected by each of the 32 possible input patterns. Also find all the minimum sets of test patterns that will discover every such fault (unless it's not detectable).

greedy algorithm
maximum independent set
contiguous United States
radio coloring
$L(2,1)$ labeling
Roberts
channel assignment
contiguous USA
square grid
grid graphs
triangular grid
simplex graph
n-cube
multiply
Dadda
Tseytin encoding
ternary operations
full adder
median operation
exclusive or, ternary
palindromic
Maximum ones
multiplications
palindrome
gates
wires
fanout gates
fanout gate
single-stuck-at faults

50. [*24*] Demonstrate Larrabee's method of representing stuck-at faults by describing the clauses that characterize test patterns for the fault "x_2^1 stuck at 1" in Fig. 34. (This is the wire that splits off of x_2 and feeds into x_2^3 and x_2^4, then to b_2 and b_3; see Table 1.)

51. [*40*] Study the behavior of SAT solvers on the problem of finding a small number of test patterns for all of the detectable single-stuck-at faults of the circuit *prod*(32, 32). Can a complete set of patterns for this large circuit be discovered "automatically" (without relying on number theory)?

52. [*15*] What clauses correspond to (29) and (30) when the *second* case on the left of Table 2, $f(1,0,1,0,\ldots,1)=1$, is taken into account?

▸ **53.** [*M20*] The numbers in Table 2 are definitely nonrandom. Can you see why?

▸ **54.** [*23*] Extend Table 2 using the rule in the previous exercise. How many rows are needed before $f(x)$ has no M-term representation in DNF, when $M = 3$, 4, and 5?

55. [*21*] Find an equation analogous to (27) that is consistent with Table 2 and has every variable complemented. (Thus the resulting function is monotone decreasing.)

▸ **56.** [*22*] Equation (27) exhibits a function matching Table 2 that depends on only 8 of the 20 variables. Use a SAT solver to show that we can actually find a suitable f that depends on only *five* of the x_j.

▸ **57.** [*29*] Combining the previous exercise with the methods of Section 7.1.2, exhibit a function f for Table 2 that can be evaluated with only six Boolean operations(!).

▸ **58.** [*20*] Discuss adding the clauses $\bar{p}_{i,j} \vee \bar{q}_{i,j}$ to (29), (30), and (31).

59. [*M20*] Compute the exact probability that $\hat{f}(x)$ in (32) differs from $f(x)$ in (27).

60. [*24*] Experiment with the problem of learning $f(x)$ in (27) from training sets of sizes 32 and 64. Use a SAT solver to find a conjectured function, $\hat{f}(x)$; then use BDD methods to determine the probability that this $\hat{f}(x)$ differs from $f(x)$ for random x.

61. [*20*] Explain how to test when a set of clauses generated from a training set via (29)–(31) is satisfiable only by the function $f(x)$ in (27).

62. [*23*] Try to learn a secret small-DNF function with N-bit training sets $x^{(0)}$, $x^{(1)}$, $x^{(2)}$, $\ldots$, where $x^{(0)}$ is random but each bit of $x^{(k)} \oplus x^{(k-1)}$ for $k > 0$ is 1 with probability p. (Thus, if p is small, successive data points will tend to be near each other.) Do such sets turn out to be more efficient in practice than the purely random ones that arise for $p = 1/2$?

▸ **63.** [*20*] Given an n-network $\alpha = [i_1 : j_1][i_2 : j_2] \ldots [i_r : j_r]$, as defined in the exercises for Section 5.3.4, explain how to use a SAT solver to test whether or not α is a sorting network. *Hint:* Use Theorem 5.3.4Z.

64. [*26*] The exact minimum time $\hat{T}(n)$ of a sorting network for n elements is a famous unsolved problem, and the fact that $\hat{T}(9) = 7$ was first established in 1987 by running a highly optimized program for many hours on a Cray 2 supercomputer.

Show that this result can now be proved with a SAT solver in less than a second(!).

▸ **65.** [*28*] Describe encodings of the Life transition function (35) into clauses.

a) Use only the variables x'_{ij} and x_{ij}.

b) Use auxiliary variables as in the Bailleux and Boufkhad encoding (20)–(21), sharing intermediate results between neighboring cells as discussed in the text.

66. [*24*] Use a SAT solver to find short counterparts to Fig. 35 in which (a) $X_1 =$ **LIFE**; (b) $X_2 =$ **LIFE**. In each case X_0 should have the smallest possible number of live cells.

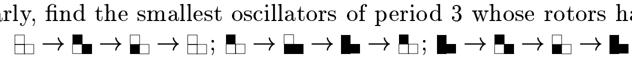
Larrabee
automatic test pattern gen
single-stuck-at faults
prod
number theory
monotone decreasing
depends on
BDD
training sets
comparator modules
sorting network
Cray 2
Life
Bailleux
Boufkhad

67. [*24*] Find a mobile chessboard path $X_0 \to X_1 \to \ldots \to X_{21}$ with no more than five cells alive in each X_t. (The glider in (37) leaves the board after X_{20}.) How about X_{22}?

68. [*39*] Find a maximum-length mobile path in which 6 to 10 cells are always alive.

69. [*23*] Find all (a) still lifes and (b) oscillators of period > 1 that live in a 4×4 board.

70. [*21*] The live cells of an oscillator are divided into a *rotor* (those that change) and a *stator* (those that stay alive).

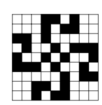
a) Show that the rotor cannot be just a single cell.
b) Find the smallest example of an oscillator whose rotor is .
c) Similarly, find the smallest oscillators of period 3 whose rotors have the following forms: .

▸ **71.** [*22*] When looking for sequences of Life transition on a square grid, an asymmetrical solution will appear in eight different forms, because the grid has eight different symmetries. Furthermore, an asymmetrical periodic solution will appear in $8r$ different forms, if r is the length of the period.

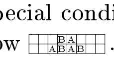
Explain how to add further clauses so that essentially equivalent solutions will occur only once: Only "canonical forms" will satisfy the conditions.

72. [*28*] Oscillators of period 3 are particularly intriguing, because Life seems so inherently binary.

a) What are the smallest such oscillators (in terms of bounding box)?
b) Find period-3 oscillators of sizes $9 \times n$ and $10 \times n$, with n odd, that have "fourfold symmetry": The patterns are unchanged after left-right and/or up-down reflection. (Such patterns are not only pleasant to look at, they also are much easier to find, because we need only consider about one-fourth as many variables.)
c) What period-3 oscillators with fourfold symmetry have the most possible live cells, on grids of sizes 15×15, 15×16, and 16×16?
d) The period-3 oscillator shown here has another kind of four-way symmetry, because it's unchanged after 90° rotation. (It was discovered in 1972 by Robert Wainwright, who called it "snake dance" because its stator involves four snakes.) What period-3 oscillators with 90° symmetry have the most possible live cells, on grids of sizes 15×15 and 16×16?



▸ **73.** [*21*] (*Mobile flipflops.*) An oscillator of period 2 is called a *flipflop*, and the Life patterns of mobile flipflops are particularly appealing: Each cell is either blank (dead at every time t) or type A (alive when t is even) or type B (alive when t is odd). Every nonblank cell (i) has exactly three neighbors of the other type, and (ii) doesn't have exactly two or three neighbors of the same type.

a) The blank cells of a mobile flipflop also satisfy a special condition. What is it?
b) Find a mobile flipflop on an 8×8 grid, with top row .
c) Find patterns that are mobile flipflops on $m \times n$ toruses for various m and n. (Thus, if replicated indefinitely, each one will tile the plane with an infinite mobile flipflop.) *Hint:* One solution has no blank cells whatsoever; another has blank cells like a checkerboard.

74. [*M28*] Continuing the previous exercise, prove that no nonblank cell of a finite mobile flipflop has more than one neighbor of its own type. (This fact greatly speeds up the search for finite mobile flipflops.) Can two type A cells be diagonally adjacent?

75. [*M22*] (Stephen Silver, 2000.) Show that a finite, mobile oscillator of period $p \geq 3$ must have some cell that is alive more than once during the cycle.

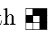
mobile
chessboard
glider
oscillator
rotor
stator
grid
symmetries
canonical forms
fourfold symmetry
reflection
rotational symmetry
chiral symmetry [rotation but not reflection)
Wainwright
snake dance
Mobile flipflops
oscillator
flipflop
toruses
Silver

76. [*41*] Construct a mobile Life oscillator of period 3.

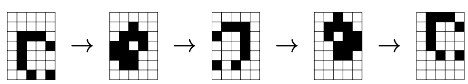
77. [*20*] "Step X_{-1}," which precedes X_0 in (38), has the glider configuration  instead of . What conditions on the still life X_5 will ensure that state X_0 is indeed reached? (We don't want digestion to begin prematurely.)

78. [*21*] Find a solution to the four-step eater problem in (38) that works on a $7 \times n$ grid, for some n, instead of 8×8.

79. [*23*] What happens if the glider meets the eater of (39) in its opposite phase (namely  instead of )?

80. [*21*] To counteract the problem in the previous exercise, find an eater that is symmetrical when reflected about a diagonal, so that it eats both  and . (You'll have to go larger than 8×8, and you'll have to wait longer for digestion.)

81. [*21*] Conway discovered a remarkable "spaceship," where X_4 is X_0 shifted up 2:

$X_0 =$  $= X_4.$

Is there a left-right symmetrical still life that will eat such spaceships?

▸ **82.** [*22*] (*Light speed.*) Imagine Life on an infinite plane, with all cells dead at time 0 except in the lower left quadrant. More precisely, suppose $X_t = (x_{tij})$ is defined for all $t \geq 0$ and all integers $-\infty < i, j < +\infty$, and that $x_{0ij} = 0$ whenever $i > 0$ or $j > 0$.

a) Prove that $x_{tij} = 0$ whenever $0 \leq t < \max(i, j)$.
b) Furthermore $x_{tij} = 0$ when $0 \leq -i \leq j$ and $0 \leq t < i + 2j$.
c) And $x_{tij} = 0$ for $0 \leq t < 2i + 2j$, if $i \geq 0$ and $j \geq 0$. *Hint:* If $x_{tij} = 0$ whenever $i \geq -j$, prove that $x_{tij} = 0$ whenever $i > -j$.

83. [*21*] According to the previous exercise, the earliest possible time that cell (i, j) can become alive, if all initial life is confined to the lower left quadrant of the plane, is at least

$$f(i, j) = i[i \geq 0] + j[j \geq 0] + (i + j)[i + j \geq 0].$$

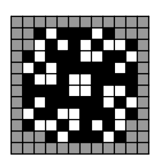
For example, when $|i| \leq 5$ and $|j| \leq 5$ the values of $f(i, j)$ are shown at the right.

5	6	7	8	9	10	12	14	16	18	20
4	4	5	6	7	8	10	12	14	16	18
3	3	3	4	5	6	8	10	12	14	16
2	2	2	2	3	4	6	8	10	12	14
1	1	1	1	1	2	4	6	8	10	12
0	0	0	0	0	0	2	4	6	8	10
0	0	0	0	0	0	1	3	5	7	9
0	0	0	0	0	0	1	2	4	6	8
0	0	0	0	0	0	1	2	3	5	7
0	0	0	0	0	0	1	2	3	4	6
0	0	0	0	0	0	1	2	3	4	5

Let $f^*(i, j)$ be the actual minimum time at which cell (i, j) can be alive, for some such initial state. Devise a set of clauses by which a SAT solver can test whether or not $f^*(i_0, j_0) = f(i_0, j_0)$, given i_0 and j_0. (Such clauses make interesting benchmark tests.)

84. [*33*] Prove that $f^*(i, j) = f(i, j)$ in the following cases when $j > 0$: (a) $i = j$, $i = j + 1$, and $i = j - 1$. (b) $i = 0$ and $i = -1$. (c) $i = 1 - j$. (d) $i = j - 2$. (e) $i = -2$.

▸ **85.** [*39*] A *Garden of Eden* is a state of Life that has no predecessor.



a) If the pattern of 92 cells illustrated here occurs anywhere within a bitmap X, verify that X is a Garden of Eden. (The gray cells can be either dead or alive.)
b) This "orphan" pattern, found with a SAT solver's help, is the smallest that is currently known. Can you imagine how it was discovered?

86. [*M23*] How many Life predecessors does a random 10×10 bitmap have, on average?

87. [*21*] Explain why the clauses (42) represent Alice and Bob's programs (40), and give a general recipe for converting such programs into equivalent sets of clauses.

Conway
spaceship
Light speed
benchmark tests
Garden of Eden
bitmap
orphan
Alice and Bob
mutual exclusion–

88. [*18*] Satisfy (41) and (42) for $0 \le t < 6$, and the 20×6 additional binary clauses that exclude multiple states, along with the "embarrassing" unit clauses $(\mathrm{A3}_6) \wedge (\mathrm{B3}_6)$.

89. [*21*] Here's a mutual-exclusion protocol once recommended in 1966. Does it work?

A0. Maybe go to A1.	B0. Maybe go to B1.
A1. Set $a \leftarrow 1$, go to A2.	B1. Set $b \leftarrow 1$, go to B2.
A2. If l go to A3, else to A5.	B2. If l go to B5, else to B3.
A3. If b go to A3, else to A4.	B3. If a go to B3, else to B4.
A4. Set $l \leftarrow 0$, go to A2.	B4. Set $l \leftarrow 1$, go to B2.
A5. Critical, go to A6.	B5. Critical, go to B6.
A6. Set $a \leftarrow 0$, go to A0.	B6. Set $b \leftarrow 0$, go to B0.

90. [*20*] Show that (43), (45), and (46) permit starvation, by satisfying (47) and (48).

91. [*M21*] Formally speaking, Alice is said to "starve" if there is (i) an infinite sequence of transitions $X_0 \to X_1 \to \cdots$ starting from the initial state X_0, and (ii) an infinite sequence $@_0, @_1, \ldots$ of Boolean "bumps" that changes infinitely often, such that (iii) Alice is in a "maybe" or "critical" state only a finite number of times. Prove that this can happen if and only if there is a starvation cycle (47) as discussed in the text.

92. [*20*] Suggest $O(r^2)$ clauses with which we can determine whether or not a mutual exclusion protocol permits a path $X_0 \to X_1 \to \cdots \to X_r$ of *distinct* states.

93. [*20*] What clauses correspond to the term $\neg\Phi(X')$ in (51)?

▸ **94.** [*21*] Suppose we know that $(X_0 \to X_1 \to \cdots \to X_r) \wedge \neg\Phi(X_r)$ is unsatisfiable for $0 \le r \le k$. What clauses will guarantee that Φ is invariant? (The case $k = 1$ is (51).)

95. [*20*] Using invariants like (50), prove that (45) and (46) provide mutual exclusion.

96. [*22*] Find all solutions to (52) when $r = 2$. Also illustrate the fact that invariants are extremely helpful, by finding a solution with distinct states $X_0, X_1, \ldots, X_r$ and with r substantially greater than 2, if the clauses involving Φ are removed.

97. [*20*] Can states A6 and B6 occur simultaneously in Peterson's protocol (49)?

▸ **98.** [*M23*] This exercise is about proving the nonexistence of starvation cycles (47).

a) A cycle of states is called "pure" if one of the players is never bumped, and "simple" if no state is repeated. Prove that the shortest impure cycle, if any, is either simple or consists of two simple pure cycles that share a common state.

b) If Alice is starved by some cycle with protocol (49), we know that she is never in states A0 or A5 within the cycle. Show that she can't be in A1, A2, or A6 either.

c) Construct clauses to test whether there exist states $X_0 \to X_1 \to \cdots \to X_r$, with X_0 arbitrary, such that $(X_0 X_1 \ldots X_{k-1})$ is a starvation cycle for some $k \le r$.

d) Therefore we can conclude that (49) is starvation-free without much extra work.

99. [*25*] Th. Dekker devised the first correct mutual-exclusion protocol in 1965:

A0. Maybe go to A1.	B0. Maybe go to B1.
A1. Set $a \leftarrow 1$, go to A2.	B1. Set $b \leftarrow 1$, go to B2.
A2. If b go to A3, else to A6.	B2. If a go to B3, else to B6.
A3. If l go to A4, else to A2.	B3. If l go to B2, else to B4.
A4. Set $a \leftarrow 0$, go to A5.	B4. Set $b \leftarrow 0$, go to B5.
A5. If l go to A5, else to A1.	B5. If l go to B1, else to B5.
A6. Critical, go to A7.	B6. Critical, go to B7.
A7. Set $l \leftarrow 1$, go to A8.	B7. Set $l \leftarrow 0$, go to B8.
A8. Set $a \leftarrow 0$, go to A0.	B8. Set $b \leftarrow 0$, go to B0.

Use bounded model checking to verify its correctness.

mutual-exclusion protocol
starvation
initial state
starvation cycle
simple path
invariant
mutual exclusion
Peterson
starvation cycles
pure cycle
simple cycle
Dekker

simultaneous write/write
simultaneous read/write
nondeterministically
flickering
bishops
tomographically balanced
Basket weavers
gipatsi patterns
pixels, rotated
grid, rotated

100. [*22*] Show that the following protocol can starve one player but not the other:

	B0. Maybe go to B1.
A0. Maybe go to A1.	B1. Set $b \leftarrow 1$, go to B2.
A1. Set $a \leftarrow 1$, go to A2.	B2. If a go to B3, else to B5.
A2. If b go to A2, else to A3.	B3. Set $b \leftarrow 0$, go to B4.
A3. Critical, go to A4.	B4. If a go to B4, else to B1.
A4. Set $a \leftarrow 0$, go to A0.	B5. Critical, go to B6.
	B6. Set $b \leftarrow 0$, go to B0.

▸ **101.** [*31*] Protocol (49) has the potential defect that Alice and Bob might both be trying to set the value of l at the same time. Design a mutual-exclusion protocol in which each of them controls two binary signals, visible to the other. *Hint:* The method of the previous exercise can be enclosed in another protocol.

102. [*22*] If Alice is setting a variable at the same time that Bob is trying to read it, we might want to consider a more stringent model under which he sees either 0 or 1, nondeterministically. (And if he looks k times before she moves to the next step, he might see 2^k possible sequences of bits.) Explain how to handle this model of "flickering" variables by modifying the clauses of exercise 87.

103. [*18*] (Do this exercise *by hand*, it's fun!) Find the 7×21 image whose tomographic sums are $(r_1,\ldots,r_7) = (1,0,13,6,12,7,19)$; $(c_1,\ldots,c_{21}) = (4,3,3,4,1,6,1,3,3,3,5,1,1,5,1,5,1,5,1,1,1)$; $(a_1,\ldots,a_{27}) = (0,0,1,2,2,3,2,3,3,2,3,3,4,3,2,3,3,3,4,3,2,2,1,1,1,1,1)$; $(b_1,\ldots,b_{27}) = (0,0,0,0,0,1,3,3,4,3,2,2,2,3,3,4,2,3,3,3,3,3,4,3,2,1,1)$.

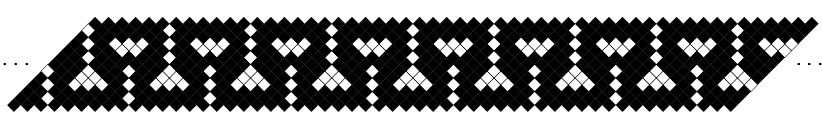
104. [*M21*] For which m and n is it possible to satisfy the digital tomography problem with $a_d = b_d = 1$ for $0 < d < m+n$? (Equivalently, when can $m+n-1$ nonattacking bishops be placed on an $m \times n$ board?)

▸ **105.** [*M28*] A matrix whose entries are $\{-1,0,+1\}$ is *tomographically balanced* if its row, column, and diagonal sums are all zero. Two binary images $X = (x_{ij})$ and $X' = (x'_{ij})$ clearly have the same row, column, and diagonal sums if and only if $X - X'$ is tomographically balanced.

a) Suppose Y is tomographically balanced and has m rows, n columns, and t occurrences of $+1$. How many $m \times n$ binary matrices X and X' satisfy $X - X' = Y$?
b) Express the condition "Y is tomographically balanced" in terms of clauses, with the values $\{-1,0,+1\}$ represented respectively by the 2-bit codes $\{10,00,01\}$.
c) Count the number $T(m,n)$ of tomographically balanced matrices, for $m,n \le 8$.
d) How many such matrices have exactly four occurrences of $+1$?
e) At most how many $+1$s can a $2n \times 2n$ tomographically balanced matrix have?
f) True or false: The positions of the $+1$s determine the positions of the -1s.

106. [*M20*] Determine a generous upper bound on the possible number of different sets of input data $\{r_i, c_j, a_d, b_d\}$ that might be given to a 25×30 digital tomography problem, by assuming that each of those sums independently has any of its possible values. How does this bound compare to 2^{750}?

▸ **107.** [*22*] Basket weavers from the Tonga culture of Inhambane, Mozambique, have developed appealing periodic designs called "gipatsi patterns" such as this:

...  ...

(Notice that an ordinary pixel grid has been rotated by 45°.) Formally speaking, a gipatsi pattern of period p and width n is a $p \times n$ binary matrix $(x_{i,j})$ in which we have

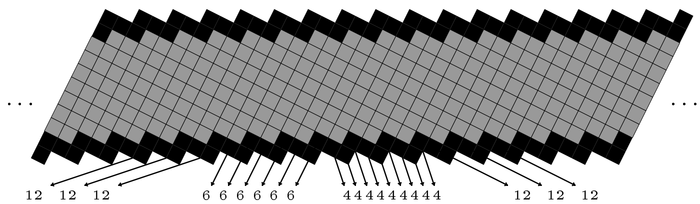
$x_{i,1} = x_{i,n} = 1$ for $1 \le i \le p$. Row i of the matrix is to be shifted right by $i-1$ places in the actual pattern. The example above has $p = 6$, $n = 13$, and the first row of its matrix is 1111101111101. Such a pattern has row sums $r_i = \sum_{j=1}^{n} x_{i,j}$ for $1 \le i \le p$ and column sums $c_j = \sum_{i=1}^{p} x_{i,j}$ for $1 \le j \le n$, as usual. By analogy with (53), it also has

$$a_d = \sum_{i+j \equiv d \,(\text{modulo}\, p)} x_{i,j}, \quad 1 \le d \le p; \qquad b_d = \sum_{2i+j \equiv d \,(\text{modulo}\, 2p)} x_{i,j}, \quad 1 \le d \le 2p.$$

a) What are the tomographic parameters r_i, c_j, a_d, and b_d in the example pattern?
b) Do any other gipatsi patterns have the same parameters?

108. [*23*] The column sums c_j in the previous exercise are somewhat artificial, because they count black pixels in only a small part of an infinite line. If we rotate the grid at a different angle, however, we can obtain infinite periodic patterns for which each of Fig. 36's four directions encounters only a finite number of pixels.

Design a pattern of period 6 in which parallel lines always have equal tomographic projections, by changing each of the gray pixels in the following diagram to either white or black:



▸ **109.** [*20*] Explain how to find the lexicographically smallest solution $x_1 \ldots x_n$ to a satisfiability problem, using a SAT solver repeatedly. (See Fig. 37(a).)

110. [*19*] What are the lexicographically (first, last) solutions to *waerden*$(3, 10; 96)$?

111. [*40*] The lexicographically first and last solutions to the "Cheshire Tom" problem in Fig. 37 are based on the top-to-bottom-and-left-to-right ordering of pixels. Experiment with other pixel orderings — for example, try bottom-to-top-and-right-to-left.

112. [*46*] Exactly how many solutions does the tomography problem of Fig. 36 have?

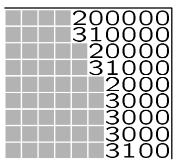
▸ **113.** [*30*] Prove that the digital tomography problem is NP-complete, even if the marginal sums r, c, a, b are binary: Show that an efficient algorithm to decide whether or not an $n \times n$ pixel image (x_{ij}) exists, having given 0–1 values of $r_i = \sum_j x_{ij}$, $c_j = \sum_i x_{ij}$, $a_d = \sum_{i+j=d+1} x_{ij}$, and $b_d = \sum_{i-j=d-n} x_{ij}$, could be used to solve the binary tensor contingency problem of exercise 212(a).

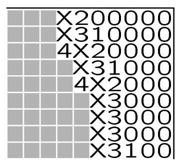
114. [*27*] Each cell (i, j) of a given rectangular grid either contains a land mine $(x_{i,j} = 1)$ or is safe $(x_{i,j} = 0)$. In the game of *Minesweeper*, you are supposed to identify all of the hidden mines, by probing locations that you hope are safe: If you decide to probe a cell with $x_{i,j} = 1$, the mine explodes and you die (at least virtually). But if $x_{i,j} = 0$ you're told the number $n_{i,j}$ of neighboring cells that contain mines, $0 \le n_{i,j} \le 8$, and you live to make another probe. By carefully considering these numeric clues, you can often continue with completely safe probes, eventually touching every mine-free cell.

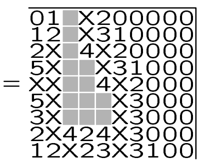
For example, suppose the hidden mines happen to match the 25×30 pattern of the Cheshire cat (Fig. 36), and you start by probing the upper right corner. That cell turns out to be safe, and you learn that $n_{1,30} = 0$; hence it's safe to probe all three neighbors of $(1, 30)$. Continuing in this vein soon leads to illustration (α) below, which depicts information about cells (i, j) for $1 \le i \le 9$ and $21 \le j \le 30$; unprobed cells are

lexicographically smallest solution
interactive SAT solving
van der Waerden numbers
waerden
digital tomography
NP-complete
0–1 matrices
binary tensor contingency problem
contingency
grid
land mine
Minesweeper

shown in gray, otherwise the value of $n_{i,j}$ appears. From this data it's easy to deduce that $x_{1,24} = x_{2,24} = x_{3,25} = x_{4,25} = \cdots = x_{9,26} = 1$; you'll never want to probe in those places, so you can mark such cells with X, arriving at state (β) since $n_{3,24} = n_{5,25} = 4$. Further progress downward to row 17, then leftward and up, leads without difficulty to state (γ). (Notice that this process is analogous to digital tomography, because you're trying to reconstruct a binary array from information about partial sums.)

$(\alpha) =$ 

; $(\beta) =$ 

; $(\gamma) =$ 

.

a) Now find safe probes for all thirteen of the cells that remain gray in (γ).

b) Exactly how much of the Cheshire cat can be revealed without making any unsafe guesses, if you're told in advance that (i) $x_{1,1} = 0$? (ii) $x_{1,30} = 0$? (iii) $x_{25,1} = 0$? (iv) $x_{25,30} = 0$? (v) all four corners are safe? *Hint:* A SAT solver can help.

115. [*25*] Empirically estimate the probability that a 9×9 game of Minesweeper, with 10 randomly placed mines, can be won with entirely safe probes after the first guess.

116. [*22*] Find examples of Life flipflops for which X and X' are tomographically equal.

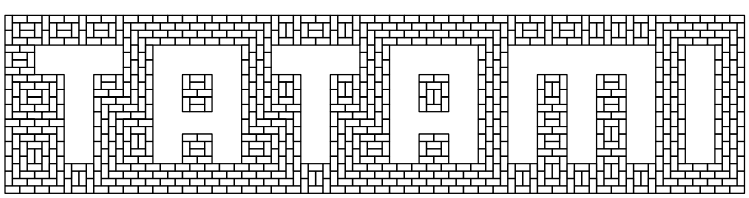
117. [*23*] Given a sequence $x = x_1 \ldots x_n$, let $\nu^{(2)}x = x_1x_2 + x_2x_3 + \cdots + x_{n-1}x_n$. (A similar sum appears in the serial correlation coefficient, 3.3.2–(23).)

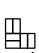
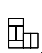
a) Show that, when x is a binary sequence, the number of runs of 1s in x can be expressed in terms of νx and $\nu^{(2)}x$.

b) Explain how to encode the condition $\nu^{(2)}x \le r$ as a set of clauses, by modifying the cardinality constraints (20)–(21) of Bailleux and Boufkhad.

c) Similarly, encode the condition $\nu^{(2)}x \ge r$.

118. [*20*] A *tatami tiling* is a covering by dominoes in which no three share a corner:



(Notice that  is disallowed, but  would be fine.) Explain how to use a SAT solver to find a tatami tiling that covers a given set of pixels, unless no such tiling exists.

119. [*18*] Let $F = \mathit{waerden}(3,3;9)$ be the 32 clauses in (9). For which literal l is the reduced formula $F \mid l$ smallest? Exhibit the resulting clauses.

120. [*M20*] True or false: $F \mid L = \{C \setminus \overline{L} \mid C \in F \text{ and } C \cap L = \emptyset\}$, if $\overline{L} = \{\bar{l} \mid l \in L\}$.

121. [*21*] Spell out the changes to the link fields in the data structures, by expanding the higher-level descriptions that appear in steps A3, A4, A7, and A8 of Algorithm A.

▸ **122.** [*21*] Modify Algorithm A so that it finds *all* satisfying assignments of the clauses.

123. [*17*] Show the contents of the internal data structures `L`, `START`, and `LINK` when Algorithm B or Algorithm D begins to process the seven clauses R' of (7).

Minesweeper
Life
flipflops
serial correlation coefficient
runs of 1s
Bailleux
Boufkhad
tatami tiling
dominoes
waerden
conditioning
reduced clauses
all solutions
satisfying assignments

▶ **124.** [*21*] Spell out the low-level link field operations that are sketched in step B3.

▶ **125.** [*20*] Modify Algorithm B so that it finds *all* satisfying assignments of the clauses.

126. [*20*] Extend the computation in (59) by one more step.

127. [*17*] What move codes $m_1 \ldots m_d$ correspond to the computation sketched in (59), just before and after backtracking occurs?

128. [*19*] Describe the entire computation by which Algorithm D proves that Rivest's clauses (6) are unsatisfiable, using a format like (59). (See Fig. 39.)

129. [*20*] In the context of Algorithm D, design a subroutine that, given a literal l, returns 1 or 0 according as l is or is not being watched in some clause whose other literals are entirely false.

130. [*22*] What low-level list processing operations are needed to "clear the watch list for $\bar{x}_k$" in step D6?

▶ **131.** [*30*] After Algorithm D exits step D3 without finding any unit clauses, it has examined the watch lists of every free variable. Therefore it could have computed the lengths of those watch lists, with little additional cost; and information about those lengths could be used to make a more informed decision about the variable that's chosen for branching in step D4. Experiment with different branching heuristics of this kind.

▶ **132.** [*22*] Theorem 7.1.1K tells us that every 2SAT problem can be solved in linear time. Is there a sequence of 2SAT clauses for which Algorithm D takes exponential time?

▶ **133.** [*25*] The size of a backtrack tree such as Fig. 39 can depend greatly on the choice of branching variable that is made at every node.

a) Find a backtrack tree for $waerden(3,3;9)$ that has the fewest possible nodes.

b) What's the *largest* backtrack tree for that problem?

134. [*22*] The `BIMP` tables used by Algorithm L are sequential lists of dynamically varying size. One attractive way to implement them is to begin with every list having capacity 4 (say); then when a list needs to become larger, its capacity can be doubled.

Adapt the buddy system (Algorithm 2.5R) to this situation. (Lists that shrink when backtracking needn't free their memory, since they're likely to grow again later.)

▶ **135.** [*16*] The literals l' in `BIMP`(l) are those for which $l \longrightarrow l'$ in the "implication digraph" of a given satisfiability problem. How can we easily find all of the literals l'' such that $l'' \longrightarrow l$, given l?

136. [*15*] What pairs will be in `TIMP`$(\bar{3})$, before and after x_5 is set to zero with respect to the clauses (9) of $waerden(3,3;9)$, assuming that we are on decision level $d = 0$?

137. [*24*] Spell out in detail the processes of (a) removing a variable X from the free list and from all pairs in `TIMP` lists (step L7 of Algorithm L), and of (b) restoring it again later (step L12). Exactly how do the data structures change?

▶ **138.** [*20*] Discuss what happens in step L9 of Algorithm L if we happen to have both $\bar{v} \in$ `BIMP`$(\bar{u})$ and $\bar{u} \in$ `BIMP`$(\bar{v})$.

139. [*25*] (*Compensation resolvents.*) If $w \in$ `BIMP`(v), the binary clause $u \vee v$ implies the binary clause $u \vee w$, because we can resolve $u \vee v$ with $\bar{v} \vee w$. Thus step L9 could exploit each new binary clause further, by appending w as well as v to `BIMP`$(\bar{u})$, for all such w. Discuss how to do this efficiently.

satisfying assignments
Rivest
watched
unit clauses
watch list
branching heuristics
2SAT
exponential time
backtrack tree, optimum
BIMP tables
sequential lists
dynamic storage allocation
buddy system
implication digraph
Compensation resolvents
resolve

140. [*21*] The FORCE, BRANCH, BACKF, and BACKI arrays in Algorithm L will obviously never contain more than n items each. Is there a fairly small upper bound on the maximum possible size of ISTACK?

141. [*18*] Algorithm L might increase ISTAMP so often that it overflows the size of the IST(l) fields. How can the mechanism of (63) avoid bugs in such a case?

142. [*24*] Algorithms A, B, and D can display their current progress by exhibiting a sequence of move codes $m_1 \ldots m_d$ such as (58) and (60); but Algorithm L has no such codes. Show that an analogous sequence $m_1 \ldots m_F$ could be printed in step L2, if desired. Use the codes of Algorithm D; but extend them to show $m_j = 6$ (or 7) if R_{j-1} is a true (or false) literal whose value was found to be forced by Algorithm X, or forced by being a unit clause in the input.

▸ **143.** [*30*] Modify Algorithm L so that it will apply to nonempty clauses of any size. Call a clause *big* if its size is greater than 2. Instead of TIMP tables, represent every big clause by 'KINX' and 'CINX' tables: Every literal l has a sequential list KINX(l) of big clause numbers; every big clause c has a sequential list CINX(c) of literals; c is in KINX(l) if and only if l is in CINX(c). The current number of active clauses containing l is indicated by KSIZE(l); the current number of active literals in c is indicated by CSIZE(c).

144. [*15*] True or false: If l doesn't appear in any clause, $h'(l) = 0.1$ in (65).

145. [*23*] Starting with $h(l) = 1$ for each of the 18 literals l in *waerden*$(3, 3; 9)$, find successively refined estimates $h'(l)$, $h''(l)$, ..., using (65) with respect to the 32 ternary clauses (9). Then, assuming that x_5 has been set false as in exercise 136, and that the resulting binary clauses 13, 19, 28, 34, 37, 46, 67, 79 have been included in the BIMP tables, do the same for the 16 literals that remain at depth $d = 1$.

146. [*25*] Suggest an alternative to (64) and (65) for use when Algorithm L has been extended to nonternary clauses as in exercise 143. (Strive for simplicity.)

147. [*05*] Evaluate $C_{\max}$ in (66) for $d = 0$, 1, 10, 20, 30, using the default C_0 and C_1.

148. [*21*] Equation (66) bounds the maximum number of candidates using a formula that depends on the current depth d, but not on the total number of free variables. The same cutoffs are used in problems with any number of variables. Why is that a reasonable strategy?

▸ **149.** [*26*] Devise a data structure that makes it convenient to tell whether a given variable x is a "participant" in Algorithm L.

150. [*21*] Continue the text's story of lookahead in *waerden*$(3, 3; 9)$: What happens at depth $d = 1$ when $l \leftarrow 7$ and $T \leftarrow 22$ (see (70)), after literal 4 has become proto true? (Assume that no double-lookahead is done.)

▸ **151.** [*26*] The dependency digraph (68) has 16 arcs, only 8 of which are captured in the subforest (69). Show that, instead of (70), we could actually list the literals l and give them offsets $o(l)$ in such a way that u appears before v in the list and has $o(u) > o(v)$ if and only if $v \longrightarrow u$ in (68). Thus we could capture all 16 dependencies via levels of truth.

152. [*22*] Give an instance of 3SAT for which no free "participants" are found in step X3, yet all clauses are satisfied. Also describe an efficient way to verify satisfaction.

153. [*17*] What's a good way to weed out unwanted candidates in step X3, if $C > C_{\max}$?

154. [*20*] Suppose we're looking ahead with just four candidate variables, $\{a, b, c, d\}$, and that they're related by three binary clauses $(a \vee \bar{b}) \wedge (a \vee \bar{c}) \wedge (c \vee \bar{d})$. Find a subforest and a sequence of truth levels to facilitate lookaheads, analogous to (69) and (70).

ISTAMP
stamping
move codes
big clause
heuristic scores
cutoffs
participant
participants

155. [*32*] Sketch an efficient way to construct the lookahead forest in step X4.

156. [*05*] Why is a pure literal a special case of an autarky?

157. [*10*] Give an example of an autarky that is not a pure literal.

158. [*15*] If l is a pure literal, will Algorithm X discover it?

159. [*M17*] True or false: (a) A is an autarky for F if and only if $F \mid A \subseteq F$. (b) If A is an autarky for F and $A' \subseteq A$, then $A \setminus A'$ is an autarky for $F \mid A'$.

160. [*18*] (*Black and white principle.*) Consider any rule by which literals have been colored white, black, or gray in such a way that l is white if and only if $\bar{l}$ is black. (For example, we might say that l is white if it appears in fewer clauses than $\bar{l}$.)

a) Suppose every clause of F that contains a white literal also contains a black literal. Prove that F is satisfiable if and only if its all-gray clauses are satisfiable.

b) Explain why this metaphor is another way to describe the notion of an autarky.

▸ **161.** [*21*] (*Black and blue principle.*) Now consider coloring literals either white, black, orange, blue, or gray, in such a way that l is white if and only if $\bar{l}$ is black, and l is orange if and only if $\bar{l}$ is blue. (Hence l is gray if and only if $\bar{l}$ is gray.) Suppose further that F is a set of clauses in which every clause containing a white literal also contains either a black literal or a blue literal (or both). Let $A = \{a_1, \ldots, a_p\}$ be the black literals and let $L = \{l_1, \ldots, l_q\}$ be the blue literals. Also let F' be the set of clauses obtained by adding p additional clauses $(\bar{l}_1 \vee \cdots \vee \bar{l}_q \vee a_j)$ to F, for $1 \le j \le p$.

a) Prove that F is satisfiable if and only if F' is satisfiable.

b) Restate and simplify that result in the case that $p = 1$.

c) Restate and simplify that result in the case that $q = 1$.

d) Restate and simplify that result in the case that $p = q = 1$. (In this special case, $(\bar{l} \vee a)$ is called a *blocked binary clause.*)

162. [*21*] Devise an efficient way to discover all of the (a) blocked binary clauses $(\bar{l} \vee a)$ and (b) size-two autarkies $A = \{a, a'\}$ of a given kSAT problem F.

▸ **163.** [*M25*] Prove that the following recursive procedure $R(F)$ will solve any n-variable 3SAT problem F with at most $O(\phi^n)$ executions of steps R1, R2, or R3:

R1. [Check easy cases.] If $F = \emptyset$, return true. If $\emptyset \in F$, return false. Otherwise let $\{l_1, \ldots, l_s\} \in F$ be a clause of minimum size s.

R2. [Check autarkies.] If $s = 1$ or if $\{l_s\}$ is an autarky, set $F \leftarrow F \mid l_s$ and return to R1. Otherwise if $\{\bar{l}_s, l_{s-1}\}$ is an autarky, set $F \leftarrow F \mid \bar{l}_s, l_{s-1}$ and return to R1.

R3. [Recurse.] If $R(F \mid l_s)$ is true, return true. Otherwise set $F \leftarrow F \mid \bar{l}_s$, $s \leftarrow s - 1$, and go back to R2. ▮

164. [*M30*] Continuing exercise 163, bound the running time when F is kSAT.

▸ **165.** [*26*] Design an algorithm to find the largest positive autarky A for a given F, namely an autarky that contains only positive literals. *Hint:* Warm up by finding the largest positive autarky for the clauses $\{12\bar{3}, 12\bar{5}, \bar{1}\bar{3}\bar{4}, 13\bar{6}, 1\bar{4}5, 156, \bar{2}35, 2\bar{4}6, 345, \bar{3}56\}$.

166. [*30*] Justify the operations of step X9. *Hint:* Prove that an autarky can be constructed, if $w = 0$ after (72) has been performed.

▸ **167.** [*21*] Justify step X11 and the similar use of X12 in step X6.

168. [*26*] Suggest a way to choose the branch literal l in step L3, based on the heuristic scores $H(l)$ that were compiled by Algorithm X in step L2. *Hint:* Experience shows that it's good to have both $H(l)$ and $H(\bar{l})$ large.

lookahead forest
pure literal
autarky
Black and white principle
Black and blue principle
blocked binary clause
kSAT
3SAT
analysis of algs
kSAT
positive autarky
positive literals
autarky
necessary assignments

▸ **169.** [*HM30*] (T. Ahmed, O. Kullmann.) Excellent results have been obtained in some problems when the branch variable in step L3 is chosen to minimize the quantity $\tau(H(l), H(\bar{l}))$, where $\tau(a, b)$ is the positive solution to $\tau^{-a} + \tau^{-b} = 1$. (For example, $\tau(1, 2) = \phi \approx 1.62$ and $\tau(\sqrt{2}, \sqrt{2}) = 2^{1/\sqrt{2}} \approx 1.63$, so we prefer $(1, 2)$ to $(\sqrt{2}, \sqrt{2})$.) Given a list of pairs of positive numbers (a_1, b_1), ..., (a_s, b_s), what's an efficient way to determine an index j that minimizes $\tau(a_j, b_j)$, without computing logarithms?

170. [*25*] (Marijn Heule, 2013.) Show that Algorithm L solves 2SAT in linear time.

171. [*20*] What is the purpose of `DFAIL` in Algorithm Y?

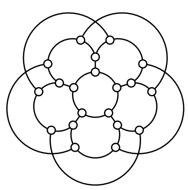
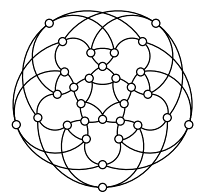
172. [*21*] Explain why '`+LO[`j`]`' appears in step Y2's formula for `DT`.

173. [*40*] Use an implementation of Algorithm L to experiment with random 3SAT problems such as *rand*$(3, 2062, 500, 314)$. Examine the effects of such things as (i) disabling double lookahead; (ii) disabling "wraparound," by changing the cases $j = S$ and $\hat{\jmath} = S$ in X7 and Y4 so that they simply go to X6 and Y3; (iii) disabling the lookahead forest, by letting all candidate literals have null `PARENT`; (iv) disabling compensation resolvents in step L9; (v) disabling "windfalls" in (72); (vi) branching on a *random* free candidate l in L3, instead of using the H scores as in exercise 168; or (vii) disabling all lookahead entirely as in "Algorithm L^0."

174. [*15*] What's an easy way to accomplish (i) in the previous exercise?

175. [*32*] When Algorithm L is extended to nonternary clauses as in exercise 143, how should Algorithms X and Y also change? (Instead of using (64) and (65) to compute a heuristic for preselection, use the much simpler formula in answer 146. And instead of using $h(u)h(v)$ in (67) to estimate the weight of a ternary clause that will be reduced to binary, consider a simulated reduced clause of size $s \ge 2$ to have weight $K_s \approx \gamma^{s-2}$, where γ is a constant (typically 0.2).)

176. [*M25*] The "flower snark" J_q is a cubic graph with $4q$ vertices t_j, u_j, v_j, w_j, and $6q$ edges t_j — t_{j+1}, t_j — u_j, u_j — v_j, u_j — w_j, v_j — w_{j+1}, w_j — v_{j+1}, for $1 \le j \le q$, with subscripts treated modulo q. Here, for example, are J_5 and its line graph $L(J_5)$:

J_5 =  ; $L(J_5)$ =  .

a) Give labels a_j, b_j, c_j, d_j, e_j, and f_j to the edges of J_q, for $1 \le j \le q$. (Thus a_j denotes t_j — t_{j+1} and b_j denotes t_j — u_j, etc.) What are the edges of $L(J_q)$?

b) Show that $\chi(J_q) = 2$ and $\chi(L(J_q)) = 3$ when q is even.

c) Show that $\chi(J_q) = 3$ and $\chi(L(J_q)) = 4$ when q is odd. *Note:* Let *fsnark*(q) denote the clauses (15) and (16) that correspond to 3-coloring $L(J_q)$, together with $b_{1,1} \wedge c_{1,2} \wedge d_{1,3}$ to set the colors of (b_1, c_1, d_1) to $(1, 2, 3)$. Also let *fsnark*$'(q)$ be *fsnark*(q) augmented by (17). These clauses make excellent benchmark tests for SAT solvers.

177. [*HM26*] Let I_q be the number of independent sets of the flower snark line graph $L(J_q)$. Compute I_q for $1 \le q \le 8$, and determine the asymptotic growth rate.

▸ **178.** [*M23*] When Algorithm B is presented with the unsatisfiable clauses *fsnark*(q) of exercise 176, with q odd, its speed depends critically on the ordering of the variables.

Ahmed
Kullmann
τ function
Heule
2SAT
DFAIL
random 3SAT
rand
pi
lookahead forest
compensation resolvents
windfalls
Algorithm L^0
heuristic
preselection
flower snark
cubic graph
trivalent graph, see cubic
line graph
chromatic number χ
fsnark
benchmark tests
independent sets
asymptotic

Show that the running time is $\Theta(2^q)$ when the variables are considered in the order

$$a_{1,1}a_{1,2}a_{1,3}b_{1,1}b_{1,2}b_{1,3}c_{1,1}c_{1,2}c_{1,3}d_{1,1}d_{1,2}d_{1,3}e_{1,1}e_{1,2}e_{1,3}f_{1,1}f_{1,2}f_{1,3}a_{2,1}a_{2,2}a_{2,3}\ldots;$$

but much, much more time is needed when the order is

$$a_{1,1}b_{1,1}c_{1,1}d_{1,1}e_{1,1}f_{1,1}a_{2,1}b_{2,1}c_{2,1}d_{2,1}e_{2,1}f_{2,1}\ldots a_{q,1}b_{q,1}c_{q,1}d_{q,1}e_{q,1}f_{q,1}a_{1,2}b_{1,2}c_{1,2}\ldots.$$

179. [*25*] Show that there are exactly 4380 ways to fill the 32 cells of the 5-cube with eight 4-element subcubes. For example, one such way is to use the subcubes 000**, 001**, ..., 111**, in the notation of 7.1.1–(29); a more interesting way is to use

$$0{*}0{*}0,\quad 1{*}0{*}0,\quad {*}{*}001,\quad {*}{*}110,\quad {*}010{*},\quad {*}110{*},\quad 0{*}{*}11,\quad 1{*}{*}11.$$

What does this fact tell you about the value of q_8 in Fig. 40?

▸ **180.** [*25*] Explain how to use BDDs to compute the numbers Q_m that underlie Fig. 40. What is $\max_{0\le m\le 80} Q_m$?

▸ **181.** [*25*] Extend the idea of the previous exercise so that it is possible to determine the probability distributions T_m of Fig. 41.

182. [*M16*] For which values of m in Fig. 41 does T_m have a constant value?

183. [*M30*] Discuss the relation between Figs. 42 and 43.

184. [*M20*] Why does (77) characterize the relation between $\hat q_m$ and q_m?

185. [*M20*] Use (77) to prove the intuitively obvious fact that $\hat q_m \ge q_m$.

186. [*M21*] Use (77) to reduce $\sum_m \hat q_m$ and $\sum_m (2m+1)\hat q_m$ to (78) and (79).

187. [*M20*] Analyze random satisfiability in the case $k = n$: What are $S_{n,n}$ and $\widehat S_{n,n}$?

▸ **188.** [*HM25*] Analyze random 1SAT, the case $k = 1$: What are $S_{1,n}$ and $\widehat S_{1,n}$?

189. [*27*] Apply BDD methods to random 3SAT problems on 50 variables. What is the approximate BDD size after m distinct clauses have been ANDed together, as m grows?

190. [*M20*] Exhibit a Boolean function of 4 variables that can't be expressed in 3CNF. (No auxiliary variables are allowed: Only x_1, x_2, x_3, and x_4 may appear.)

191. [*M25*] How many Boolean functions of 4 variables *can* be expressed in 3CNF?

▸ **192.** [*HM21*] Another way to model satisfiability when there are N equally likely clauses is to study $S(p)$, the probability of satisfiability when each clause is independently present with probability p.

a) Express $S(p)$ in terms of the numbers $Q_m = \binom{N}{m} q_m$.

b) Assign uniform random numbers in $[0\,..\,1)$ to each clause; then at time t, for $0 \le t \le N$, consider all clauses that have been assigned a number less than t/N. (Approximately t clauses will therefore be selected, when N is large.) Show that $\overline{S}_{k,n} = \int_0^N S_{k,n}(t/N)\,dt$, the expected amount of time during which the chosen clauses remain satisfiable, is very similar to the satisfiability threshold $S_{k,n}$ of (76).

193. [*HM48*] Determine the satisfiability threshold (81) of random 3SAT. Is it true that $\liminf_{n\to\infty} S_{3,n}/n = \limsup_{n\to\infty} S_{3,n}/n$? If so, is the limit ≈ 4.2667?

194. [*HM49*] If $\alpha < \liminf_{n\to\infty} S_{3,n}/n$, is there a polynomial-time algorithm that is able to satisfy $\lfloor \alpha n\rfloor$ random 3SAT clauses with probability $\ge \delta$, for some $\delta > 0$?

195. [*HM21*] (J. Franco and M. Paull, 1983.) Use the first moment principle MPR–(21) to prove that $\lfloor (2^k \ln 2)n\rfloor$ random kSAT clauses are almost always unsatisfiable. *Hint:* Let $X = \sum_x [x \text{ satisfies all clauses}]$, summed over all 2^n binary vectors $x = x_1 \ldots x_n$.

subcubes
n-cube
BDD
1SAT
BDD
3SAT–
3CNF
auxiliary variables
random satisfiability
stopping time
Franco
Paull
kSAT
first moment principle

▸ **196.** [*HM25*] (D. B. Wilson.) A clause of a satisfiability problem is "easy" if it contains one or more variables that don't appear in any other clauses. Prove that, with probability $1 - O(n^{-2\epsilon})$, a kSAT problem that has $m = \lfloor \alpha n \rfloor$ random clauses contains $(1-(1-e^{-k\alpha})^k)m + O(n^{1/2+\epsilon})$ easy ones. (For example, about $0.000035n$ of the $4.27n$ clauses in a random 3SAT problem near the threshold will be easy.)

197. [*HM21*] Prove that the quotient $q(a,b,A,B,n) = \binom{(a+b)n}{an}\binom{(A+B)n}{An}\big/\binom{(a+b+A+B)n}{(a+A)n}$ is $O(n^{-1/2})$ as $n \to \infty$, if $a, b, A, B > 0$.

▸ **198.** [*HM30*] Use exercises 196 and 197 to show that the phase transition in Fig. 46 is not extremely abrupt: If $S_3(m,n) > \frac{2}{3}$ and $S_3(m',n) < \frac{1}{3}$, prove that $m' = m + \Omega(\sqrt{n})$.

199. [*M21*] Let $p(t,m,N)$ be the probability that t specified letters each occur at least once within a random m-letter word on an N-letter alphabet.

a) Prove that $p(t,m,N) \le m^{\underline{t}}/N^t$.
b) Derive the exact formula $p(t,m,N) = \sum_k \binom{t}{k}(-1)^k(N-k)^m/N^m$.
c) And $p(t,m,N)/t! = \left\{{t \atop t}\right\}\binom{m}{t}/N^t - \left\{{t+1 \atop t}\right\}\binom{m}{t+1}/N^{t+1} + \left\{{t+2 \atop t}\right\}\binom{m}{t+2}/N^{t+2} - \cdots$.

▸ **200.** [*M21*] Complete the text's proof of (84) when $c < 1$:

a) Show that every unsatisfiable 2SAT formula contains clauses of a snare.
b) Conversely, are the clauses of a snare always unsatisfiable?
c) Verify the inequality (89). *Hint:* See exercise 199.

201. [*HM29*] The t-snake clauses specified by a chain $(l_1, \ldots, l_{2t-1})$ can be written $(\bar{l}_i \vee l_{i+1})$ for $0 \le i < 2t$, where $l_0 = \bar{l}_t$ and subscripts are treated mod $2t$.

a) Describe all ways to set two of the l's so that $(\bar{x}_1 \vee x_2)$ is one of those $2t$ clauses.
b) Similarly, set three l's in order to obtain $(\bar{x}_1 \vee x_2)$ and $(\bar{x}_2 \vee x_3)$.
c) Also set three to obtain both $(\bar{x}_0 \vee x_1)$ and $(\bar{x}_{t-1} \vee x_t)$; here $\bar{x}_0 \equiv x_t$ and $t > 2$.
d) How can the clauses $(\bar{x}_i \vee x_{i+1})$ for $0 \le i < t$ all be obtained by setting t of the l's?
e) In general, let $N(q,r)$ be the number of ways to choose r of the standard clauses $(\bar{x}_i \vee x_{i+1})$, which involve exactly q of the variables $\{x_1, \ldots, x_{2t-1}\}$, and to set q values of $\{l_1, \ldots, l_{2t-1}\}$ in order to obtain the r chosen clauses. Evaluate $N(2,1)$.
f) Similarly, evaluate $N(3,2)$, $N(t,t)$, and $N(2t-1,2t)$.
g) Show that the probability p_r in (95) is $\le \sum_q N(q,r)/(2^q n^{\underline{q}})$.
h) Therefore the upper bound (96) is valid.

202. [*HM21*] This exercise amplifies the text's proof of Theorem C when $c > 1$.

a) Explain the right-hand side of Eq. (93).
b) Why does (97) follow from (95), (96), and the stated choices of t and m?

▸ **203.** [*HM33*] (K. Xu and W. Li, 2000.) Beginning with the n graph-coloring clauses (15), and optionally the $n\binom{d}{2}$ exclusion clauses (17), consider using randomly generated binary clauses instead of (16). There are mq random binary clauses, obtained as m independent sets of q clauses each, where every such set is selected by choosing distinct vertices u and v, then choosing q distinct binary clauses $(\bar{u}_i \vee \bar{v}_j)$ for $1 \le i, j \le d$. (The number of different possible sequences of random clauses is therefore exactly $\left(\binom{n}{2}\binom{d^2}{q}\right)^m$ and each sequence is equally likely.) This method of clause generation is known as "Model RB"; it generalizes random 2SAT, which is the case $d = 2$ and $q = 1$.

Suppose $d = n^\alpha$ and $q = pd^2$, where we require $\frac{1}{2} < \alpha < 1$ and $0 \le p \le \frac{1}{2}$. Also let $m = rn \ln d$. For this range of the parameters, we will prove that there is a sharp threshold of satisfiability: The clauses are unsatisfiable q.s., as $n \to \infty$, if $r \ln(1-p) + 1 < 0$; but they are satisfiable a.s. if $r \ln(1-p) + 1 > 0$.

Wilson
easy clauses
bystanders, see easy clauses
binomial coefficients
phase transition
sharp threshold
random word
Stirling subset numbers
2SAT
snare
t-snake
Xu
Li
exclusion clauses
at-most-one
Model RB
random 2SAT
threshold of satisfiability
q.s.
a.s.

Let $X(j_1,\ldots,j_n) = [\text{all clauses are satisfied when each } i\text{th variable } v \text{ has } v_{j_i} = 1]$; here $1 \le j_1,\ldots,j_n \le d$. Also let $X = \sum_{1\le j_1,\ldots,j_n\le d} X(j_1,\ldots,j_n)$. Then $X = 0$ if and only if the clauses are unsatisfiable.

a) Use the first moment principle to prove that $X = 0$ q.s. when $r\ln(1-p)+1 < 0$.

b) Find a formula for $p_s = \Pr(X(j_1,\ldots,j_n) = 1 \mid X(1,\ldots,1) = 1)$, given that exactly s of the colors $\{j_1,\ldots,j_n\}$ are equal to 1.

c) Use (b) and the conditional expectation inequality MPR–(24) to prove that $X > 0$ a.s. if

$$\sum_{s=0}^{n} \binom{n}{s} \Bigl(\frac{1}{d}\Bigr)^{s} \Bigl(1-\frac{1}{d}\Bigr)^{n-s} \Bigl(1+\frac{p}{1-p}\frac{s^2}{n^2}\Bigr)^{m} \to 1 \qquad \text{as } n\to\infty.$$

d) Letting t_s denote the term for s in that sum, prove that $\sum_{s=0}^{3n/d} t_s \approx 1$.

e) Suppose $r\ln(1-p)+1 = \epsilon > 0$, where ϵ is small. Show that the terms t_s first increase, then decrease, then increase, then decrease again, as s grows from 0 to n. *Hint:* Consider the ratio $x_s = t_{s+1}/t_s$.

f) Finally, prove that t_s is exponentially small for $3n/d \le s \le n$.

▸ **204.** [*28*] Figure 46 might suggest that 3SAT problems on n variables are always easy when there are fewer than $2n$ clauses. We shall prove, however, that *any* set of m ternary clauses on n variables can be transformed mechanically into another set of ternary clauses on $N = O(m)$ variables in which no variable occurs more than four times. The transformed problem is satisfiable if and only if the original problem was; thus it isn't any simpler, although (with at most $4N$ literals) it has at most $\frac{4}{3}N$ clauses.

a) First replace the original m clauses by m new clauses $(X_1\vee X_2\vee X_3)$, ..., $(X_{3m-2}\vee X_{3m-1}\vee X_{3m})$, on $3m$ new variables, and show how to add $3m$ clauses of size 2 so that the resulting $4m$ clauses have exactly as many solutions as the original.

b) Construct 16 unsatisfiable ternary clauses on 15 variables, where each variable occurs at most four times. *Hint:* If F and F' are sets of clauses, let $F \sqcup F'$ stand for any other set obtained from $F\cup F'$ by replacing one or more clauses C of F by $x\cup C$ and one or more clauses C' of F' by $\bar{x}\cup C'$, where x is a new variable; then $F\sqcup F'$ is unsatisfiable whenever F and F' are both unsatisfiable. For example, if $F=\{\epsilon\}$ and $F'=\{1,\bar{1}\}$, then $F\sqcup F'$ is either $\{2, 1\bar{2}, \bar{1}\bar{2}\}$ or $\{2, 1, \bar{1}\bar{2}\}$ or $\{2, 1\bar{2}, \bar{1}\}$.

c) Remove one of the clauses from solution (b) and find all solutions of the 15 clauses that remain. (At least three of the variables will have forced values.)

d) Use (a), (b), and (c) to prove the N-variable result claimed above.

205. [*26*] Construct an unsatisfiable 4SAT problem in which every variable occurs at most 5 times. *Hint:* Use the $\sqcup$ operation as in the previous exercise.

206. [*M22*] A set of clauses is *minimally unsatisfiable* if it is unsatisfiable, yet becomes satisfiable if any clause is deleted. Show that, if F and F' have no variables in common, then $F\sqcup F'$ is minimally unsatisfiable if and only if F and F' are minimally unsatisfiable.

207. [*25*] Each of the literals $\{1,\bar{1},2,\bar{2},3,\bar{3},4,\bar{4}\}$ occurs exactly thrice in the eight unsatisfiable clauses (6). Construct an unsatisfiable 3SAT problem with 15 variables in which each of the 30 literals occurs exactly *twice*. *Hint:* Consider $\{\bar{1}2, \bar{2}3, \bar{3}1, 123, \bar{1}\bar{2}\bar{3}\}$.

208. [*25*] Via exercises 204(a) and 207, show that any 3SAT problem can be transformed into an equivalent set of ternary clauses where every literal occurs just twice.

209. [*25*] (C. A. Tovey.) Prove that every kSAT formula in which no variable occurs more than k times is satisfiable. (Thus the limits on occurrences in exercises 204–208 cannot be lowered, when $k=3$ and $k=4$.) *Hint:* Use the theory of bipartite matching.

first moment principle
conditional expectation inequality
4SAT
minimally unsatisfiable
clauses per literal
Tovey
bipartite matching
matching

210. [*M36*] But the result in the previous exercise can be improved when k is large. Use the Local Lemma to show that every 7SAT problem with at most 13 occurrences of each variable is satisfiable.

211. [*30*] (R. W. Irving and M. Jerrum, 1994.) Use exercise 208 to reduce 3SAT to the problem of list coloring a grid graph of the form $K_N \,\square\, K_3$. (Hence the latter problem, which is also called *latin rectangle construction*, is NP-complete.)

212. [*32*] Continuing the previous exercise, we shall reduce grid list coloring to another interesting problem called *partial latin square construction*. Given three $n \times n$ binary matrices (r_{ik}), (c_{jk}), (p_{ij}), the task is to construct an $n \times n$ array (X_{ij}) such that X_{ij} is blank when $p_{ij} = 0$, otherwise $X_{ij} = k$ for some k with $r_{ik} = c_{jk} = 1$; furthermore the nonblank entries must be distinct in each row and column.

a) Show that this problem is symmetrical in all three coordinates: It's equivalent to constructing a binary $n \times n \times n$ tensor (x_{ijk}) such that $x_{*jk} = c_{jk}$, $x_{i*k} = r_{ik}$, and $x_{ij*} = p_{ij}$, for $1 \le i, j, k \le n$, where '$*$' denotes summing an index from 1 to n. (Therefore it is also known as the *binary $n \times n \times n$ contingency problem*, given n^2 row sums, n^2 column sums, and n^2 pile sums.)

b) A necessary condition for solution is that $c_{*k} = r_{*k}$, $c_{j*} = p_{*j}$, and $r_{i*} = p_{i*}$. Exhibit a small example where this condition is not sufficient.

c) If $M < N$, reduce $K_M \,\square\, K_N$ list coloring to the problem of $K_N \,\square\, K_N$ list coloring.

d) Finally, explain how to reduce $K_N \square K_N$ list coloring to the problem of constructing an $n \times n$ partial latin square, where $n = N + \sum_{I,J} |L(I,J)|$. *Hint:* Instead of considering integers $1 \le i, j, k \le n$, let i, j, k range over a *set* of n elements. Define $p_{ij} = 0$ for most values of i and j; also make $r_{ik} = c_{ik}$ for all i and k.

▸ **213.** [*M20*] Experience with the analyses of sorting algorithms in Chapter 5 suggests that random satisfiability problems might be modeled nicely if we assume that, in each of m independent clauses, the literals x_j and $\bar{x}_j$ occur with respective probabilities p and q, independently for $1 \le j \le n$, where $p + q \le 1$. Why is this *not* an interesting model as $n \to \infty$, when p and q are constant? *Hint:* What is the probability that $x_1 \ldots x_n = b_1 \ldots b_n$ satisfies all of the clauses, when $b_1 \ldots b_n$ is a given binary vector?

214. [*HM38*] Although the random model in the preceding exercise doesn't teach us how to solve SAT problems, it does lead to interesting mathematics: Let $0 < p < 1$ and consider the recurrence

$$T_0 = 0; \qquad T_n = n + 2\sum_{k=0}^{n-1} \binom{n}{k} p^k (1-p)^{n-k} T_k, \quad \text{for } n > 0.$$

a) Find a functional relation satisfied by $T(z) = \sum_{n=0}^{\infty} T_n z^n/n!$.

b) Deduce that we have $T(z) = ze^z \sum_{m=0}^{\infty} (2p)^m \prod_{k=0}^{m-1} (1 - e^{-p^k(1-p)z})$.

c) Hence, if $p \ne 1/2$, we can use Mellin transforms (as in the derivation of 5.2.2–(50)) to show that $T_n = C_p n^\alpha \bigl(1 + \delta(n) + O(1/n)\bigr) + n/(1-2p)$, where $\alpha = 1/\lg(1/p)$, C_p is a constant, and δ is a small "wobble" with $\delta(n) = \delta(pn)$.

▸ **215.** [*HM23*] What is the expected profile of the search tree when a simple backtrack procedure is used to find all solutions to a random 3SAT problem with m independent clauses on n variables? (There is a node on level l for every partial solution $x_1 \ldots x_l$ that doesn't contradict any of the clauses.) Compute these values when $m = 200$ and $n = 50$. Also estimate the total tree size when $m = \alpha n$, for fixed α as $n \to \infty$.

216. [*HM38*] (P. W. Purdom, Jr., and C. A. Brown.) Extend the previous exercise to a more sophisticated kind of backtracking, where all choices forced by unit clauses are

Local Lemma
7SAT
Irving
Jerrum
list coloring
grid graph
latin rectangle construction
NP-complete
grid list coloring
partial latin square construction
binary matrices
tensor
row sums
column sums
pile sums
contingency tables, 3D
random satisfiability
recurrence
generating functions
Mellin transforms
asymptotics
wobble
profile
backtrack
asymptotics
Purdom
Brown

pursued before two-way branching is done. (The "pure literal rule" is not exploited, however, because it doesn't find all solutions.) Prove that the expected tree size is greatly reduced when $m = 200$ and $n = 50$. (An upper bound is sufficient.)

217. [*20*] True or false: If A and B are arbitrary clauses that are simultaneously satisfiable, and if l is any literal, then the clause $C = (A \cup B) \setminus \{l, \bar{l}\}$ is also satisfiable. (We're thinking here of A, B, and C as *sets* of literals, not as disjunctions of literals.)

218. [*20*] Express the formula $(x \vee A) \wedge (\bar{x} \vee B)$ in terms of the ternary operator $u ?\, v \colon w$.

▶ **219.** [*M20*] Formulate a general definition of the resolution operator $C = C' \diamond C''$ that (i) agrees with the text's definition when $C' = x \vee A'$ and $C'' = \bar{x} \vee A''$; (ii) applies to *arbitrary* clauses C' and C''; (iii) has the property that $C' \wedge C''$ implies $C' \diamond C''$.

220. [*M24*] We say that clause C *subsumes* clause C', written $C \subseteq C'$, if $C' = \wp$ or if $C' \neq \wp$ and every literal of C appears in C'.

a) True or false: $C \subseteq C'$ and $C' \subseteq C''$ implies $C \subseteq C''$.

b) True or false: $(C \vee \alpha) \diamond (C' \vee \alpha') \subseteq (C \diamond C') \vee \alpha \vee \alpha'$, with $\diamond$ as in exercise 219.

c) True or false: $C' \subseteq C''$ implies $C \diamond C' \subseteq C \diamond C''$.

d) The notation $C_1, \ldots, C_m \vdash C$ means that a resolution chain $C_1, \ldots, C_{m+r}$ exists with $C_{m+r} \subseteq C$, for some $r \geq 0$. Show that we might have $C_1, \ldots, C_m \vdash C$ even though C cannot be obtained from $\{C_1, \ldots, C_m\}$ by successive resolutions (104).

e) Prove that if $C_1 \subseteq C_1', \ldots, C_m \subseteq C_m'$, and $C_1', \ldots, C_m' \vdash C$, then $C_1, \ldots, C_m \vdash C$.

f) Furthermore $C_1, \ldots, C_m \vdash C$ implies $C_1 \vee \alpha_1, \ldots, C_m \vee \alpha_m \vdash C \vee \alpha_1 \vee \cdots \vee \alpha_m$.

221. [*16*] Draw the search tree analogous to Fig. 38 that is implicitly traversed when Algorithm A is applied to the unsatisfiable clauses $\{12, 2, \bar{2}\}$. Explain why it does *not* correspond to a resolution refutation that is analogous to Fig. 48.

222. [*M30*] (Oliver Kullmann, 2000.) Prove that, for every clause C in a satisfiability problem F, there is an autarky satisfying C if and only if C cannot be used as the label of a source vertex in any resolution refutation of F.

223. [*HM40*] Step X9 deduces a binary clause that cannot be derived by resolution (see exercise 166). Prove that, nevertheless, the running time of Algorithm L on unsatisfiable input will never be less than the length of a shortest treelike refutation.

224. [*M20*] Given a resolution tree that refutes the axioms $F \,|\, \bar{x}$, show how to construct a resolution tree of the same size that either refutes the axioms F or derives the clause $\{x\}$ from F without resolving on the variable x.

▶ **225.** [*M31*] (G. S. Tseytin, 1966.) If T is any resolution tree that refutes a set of axioms F, show how to convert it to a *regular* resolution tree T_r that refutes F, where T_r is no larger than T.

226. [*M20*] If α is a node in a refutation tree, let $C(\alpha)$ be its label, and let $\|\alpha\|$ denote the number of leaves in its subtree. Show that, given a refutation tree with N leaves, the Prover can find a node with $\|\alpha\| \leq N/2^s$ for which the current assignment falsifies $C(\alpha)$, whenever the Delayer has scored s points in the Prover–Delayer game.

227. [*M27*] Given an extended binary tree, exercise 7.2.1.6–124 explains how to label each node with its Horton–Strahler number. For example, the nodes at depth 2 in Fig. 48 are labeled 1, because their children have the labels 1 and 0; the root is labeled 3.

Prove that the maximum score that the Delayer can guarantee, when playing the Prover–Delayer game for a set of unsatisfiable clauses F, is equal to the minimum possible Horton–Strahler root label in a tree refutation of F.

pure literal
if-then-else operator
Notation $C' \diamond C''$
Notation $C \subseteq C'$
Notation $F \vdash C$
resolution chain
search tree
resolution refutation
Kullmann
autarky
resolution tree
Tseytin
regular resolution
treelike resolution
refutation tree
Prover–Delayer game
Horton–Strahler number
Strahler

▶ **228.** [*M21*] Stålmarck's refutation of (99)–(101) actually obtains ϵ without using all of the axioms! Show that only about 1/3 of those clauses are sufficient for unsatisfiability.

▶ **229.** [*M21*] Continuing exercise 228, prove also that the set of clauses (99), (100′), (101) is unsatisfiable, where (100′) denotes (100) restricted to the cases $i \le k$ and $j < k$.

230. [*M22*] Show that the clauses with $i \ne j$ in the previous exercise form a *minimal* unsatisfiable set: Removing any one of them leaves a satisfiable remainder.

231. [*M30*] (Sam Buss.) Refute the clauses of exercise 229 with a resolution chain of length $O(m^3)$. *Hint:* Derive the clauses $G_{ij} = (x_{ij} \vee x_{i(j+1)} \vee \cdots \vee x_{im})$ for $1 \le i \le j \le m$.

▶ **232.** [*M28*] Prove that the clauses *fsnark*(q) of exercise 176 can be refuted by treelike resolution in $O(q^6)$ steps.

233. [*16*] Explain why (105) satisfies (104), by exhibiting $j(i)$ and $k(i)$ for $9 \le i \le 22$.

234. [*20*] Show that the Delayer can score at least m points against any Prover who tries to refute the pigeonhole clauses (106) and (107).

▶ **235.** [*30*] Refute those pigeonhole clauses with a chain of length $m(m+3)2^{m-2}$.

236. [*48*] Is the chain in the previous exercise as short as possible?

▶ **237.** [*28*] Show that a polynomial number of steps suffice to refute the pigeonhole clauses (106), (107), if the *extended resolution* trick is used to append new clauses.

238. [*HM21*] Complete the proof of Lemma B. *Hint:* Make $r \le \rho^{-b}$ when $W = b$.

▶ **239.** [*M21*] What clauses α_0 on n variables make $\|\alpha_0 \vdash \epsilon\|$ as large as possible?

▶ **240.** [*HM23*] Choose integers $f_{ij} \in \{1, \ldots, m\}$ uniformly at random, for $1 \le i \le 5$ and $0 \le j \le m$, and let G_0 be the bipartite graph with edges a_j — b_k if and only if $k \in \{f_{1j}, \ldots, f_{5j}\}$. Show that $\Pr\bigl(G_0$ satisfies the strong expansion condition (108)$\bigr) \ge 1/2$.

241. [*20*] Prove that any set of at most $m/3000$ pigeons can be matched to distinct holes, under the restricted pigeonhole constraints G_0 of Theorem B.

242. [*M20*] The pigeonhole axioms (106) and (107) are equivalent to the clauses (15) and (16) that arise if we try to color the complete graph K_{m+1} with m colors.

Suppose we include further axioms corresponding to (17), namely

$$(\bar{x}_{jk} \vee \bar{x}_{jk'}), \qquad \text{for } 0 \le j \le m \text{ and } 1 \le k < k' \le m.$$

Does Theorem B still hold, or do these additional axioms decrease the refutation width?

243. [*HM31*] (E. Ben-Sasson and A. Wigderson.) Let F be a set of $\lfloor \alpha n \rfloor$ random 3SAT clauses on n variables, where $\alpha > 1/e$ is a given constant. For any clause C on those variables, define $\mu(C) = \min\{\,|F'| \mid F' \subseteq F \text{ and } F' \vdash C\}$. Also let $V(F')$ denote the variables that occur in a given family of clauses F'.

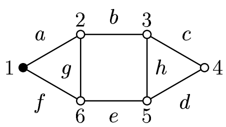
a) Prove that $|V(F')| \ge |F'|$ a.s., when $F' \subseteq F$ and $|F'| \le n/(2\alpha e^2)$.
b) Therefore either F is satisfiable or $\mu(\epsilon) > n/(2\alpha e^2)$, a.s.
c) Let $n' = n/(1000000\alpha^4)$, and assume that $n' \ge 2$. Prove that $2|V(F')| - 3|F'| \ge n'/4$ q.s., when $F' \subseteq F$ and $n'/2 \le |F'| < n'$.
d) Consequently either F is satisfiable or $w(F \vdash \epsilon) \ge n'/4$, a.s.

244. [*M20*] If A is a set of variables, let $[A]^0$ or $[A]^1$ stand for the set of all clauses that can be formed from A with an even or odd number of negative literals, respectively; each clause should involve all of the variables. (For example, $[\{1,2,3\}]^1 = \{12\bar{3}, 1\bar{2}3, \bar{1}23, \bar{1}\bar{2}\bar{3}\}$.) If A and B are disjoint, express $[A \cup B]^0$ in terms of the sets $[A]^0$, $[A]^1$, $[B]^0$, $[B]^1$.

Stålmarck
anti-maximal-element clauses
minimal unsatisfiable set
Buss
fsnark
treelike resolution
flower snark
Delayer
pigeonhole
pigeonhole
color
complete graph
exclusion clauses
Ben-Sasson
Wigderson
random 3SAT
3SAT
notation: $\mu(C)$
notation $F \vdash C$
a.s.: almost surely
q.s.: quite surely

▸ **245.** [*M27*] Let G be a connected graph whose vertices $v \in V$ have each been labeled 0 or 1, where the sum of all labels is odd. We will construct clauses on the set of variables e_{uv}, one for each edge $u \,—\, v$ in G. The axioms are $\alpha(v) = [E(v)]^{l(v)\oplus 1}$ for each $v \in V$ (see exercise 244), where $E(v) = \{e_{uv} \mid u \,—\, v\}$ and $l(v)$ is the label of v.

For example, vertex 1 of the graph below is shown as a black dot in order to indicate that $l(1) = 1$, while the other vertices appear as white dots and are labeled $l(2) = \cdots = l(6) = 0$. The graph and its axioms are

$G =$ 

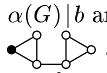
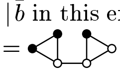
,

$$\begin{aligned} &\alpha(1) = \{af, \bar a\bar f\}, && \alpha(4) = \{c\bar d, \bar c d\},\\ &\alpha(2) = \{ab\bar g, a\bar b g, \bar a b g, \bar a\bar b\bar g\}, && \alpha(5) = \{de\bar h, d\bar e h, \bar d e h, \bar d\bar e\bar h\},\\ &\alpha(3) = \{bc\bar h, b\bar c h, \bar b c h, \bar b\bar c\bar h\}, && \alpha(6) = \{ef\bar g, e\bar f g, \bar e f g, \bar e\bar f\bar g\}.\end{aligned}$$

Notice that, when v has $d > 0$ neighbors in G, the set $\alpha(v)$ consists of 2^{d-1} clauses of size d. Furthermore, the axioms of $\alpha(v)$ are all satisfied if and only if

$$\bigoplus_{e_{uv}\in E(v)} e_{uv} = \ell(v).$$

If we sum this equation over all vertices v, mod 2, we get 0 on the left, because each edge e_{uv} occurs exactly twice (once in $E(u)$ and once in $E(v)$). But we get 1 on the right. Therefore the clauses $\alpha(G) = \bigcup_v \alpha(v)$ are unsatisfiable.

a) The axioms $\alpha(G)\,|\,b$ and $\alpha(G)\,|\,\bar b$ in this example turn out to be $\alpha(G')$ and $\alpha(G'')$, where $G' =$  and $G'' =$ . Explain what happens in general.

b) Let $\mu(C) = \min\{\,|V'| \mid V' \subseteq V$ and $\bigcup_{v\in V'} \alpha(v) \vdash C\}$, for every clause C involving the variables e_{uv}. Show that $\mu(C) = 1$ for every axiom $C \in \alpha(G)$. What is $\mu(\epsilon)$?

c) If $V' \subseteq V$, let $\partial V' = \{\, e_{uv} \mid u \in V'$ and $v \notin V'\}$. Prove that, if $|V'| = \mu(C)$, every variable of $\partial V'$ appears in C.

d) A nonbipartite cubic Ramanujan graph G on m vertices V has three edges $v \,—\, v\rho$, $v \,—\, v\sigma$, $v \,—\, v\tau$ touching each vertex, where ρ, σ, and τ are permutations with the following properties: (i) $\rho = \rho^-$ and $\tau = \sigma^-$; (ii) G is connected; (iii) If V' is any subset of s vertices, and if there are t edges between V' and $V \setminus V'$, then we have $s/(s+t) \le (s/n + 8)/9$. Prove that $w(\alpha(G) \vdash \epsilon) > m/78$.

▸ **246.** [*M28*] (G. S. Tseytin.) Given a labeled graph G with m edges, n vertices, and N unsatisfiable clauses $\alpha(G)$ as in the previous exercise, explain how to refute those clauses with $O(mn + N)$ steps of extended resolution.

247. [*18*] Apply variable elimination to just five of the six clauses (112), omitting '$1\bar 2$'.

248. [*M20*] Formally speaking, SAT is the problem of evaluating the quantified formula

$$\exists x_1 \ldots \exists x_{n-1}\, \exists x_n\, F(x_1, \ldots, x_{n-1}, x_n),$$

where F is a Boolean function given in CNF as a conjunction of clauses. Explain how to transform the CNF for F into the CNF for F' in the reduced problem

$$\exists x_1 \ldots \exists x_{n-1} F'(x_1, \ldots, x_{n-1}),\ F'(x_1, \ldots, x_{n-1}) = F(x_1, \ldots, x_{n-1}, 0) \vee F(x_1, \ldots, x_{n-1}, 1).$$

249. [*18*] Apply Algorithm I to (112) using Cook's Method IA.

250. [*25*] Since the clauses R' in (7) are satisfiable, Algorithm I might discover a solution without ever reaching step I4. Try, however, to make the choices in steps I2, I3, and I4 so that the algorithm takes as long as possible to discover a solution.

graph-based axioms
cubic
Ramanujan graph
Tseytin
extended resolution
variable elimination
quantified formula
CNF
Cook
Method IA
clause learning

▶ **251.** [*30*] Show that Algorithm I can prove the unsatisfiability of the anti-maximal-element clauses (99)–(101) by making $O(m^3)$ resolutions, if suitably clairvoyant choices are made in steps I2, I3, and I4.

252. [*M26*] Can the unsatisfiability of (99)–(101) be proved in polynomial time by repeatedly performing variable elimination and subsumption?

▶ **253.** [*18*] What are the next two clauses learned if decision '5' follows next after (114)?

254. [*16*] Given the binary clauses $\{12, \bar{1}3, 2\bar{3}, \bar{2}\bar{4}, \bar{3}4\}$, what clause will a CDCL solver learn first if it begins by deciding that 1 is true?

▶ **255.** [*20*] Construct a satisfiability problem with ternary clauses, for which a CDCL solver that is started with decision literals '1', '2', '3' on levels 1, 2, and 3 will learn the clause '45' after a conflict on level 3.

256. [*20*] How might the clause '$**$' in Table 3 have been easily learned?

▶ **257.** [*30*] (Niklas Sörensson.) A literal $\bar{l}$ is said to be *redundant*, with respect to a given clause c and the current trail, if l is in the trail and either (i) l is defined at level 0, or (ii) l is not a decision literal and every false literal in l's reason is either in c or (recursively) redundant. (This definition is stronger than the special cases by which (115) reduces to (116), because $\bar{l}$ itself needn't belong to c.) If, for example, $c = (\bar{l}' \vee \bar{b}_1 \vee \bar{b}_2 \vee \bar{b}_3 \vee \bar{b}_4)$, let the reason for b_4 be $(b_4 \vee \bar{b}_1 \vee \bar{a}_1)$, where the reason for a_1 is $(a_1 \vee \bar{b}_2 \vee \bar{a}_2)$ and the reason for a_2 is $(a_2 \vee \bar{b}_1 \vee \bar{b}_3)$. Then $\bar{b}_4$ is redundant, because $\bar{a}_2$ and $\bar{a}_1$ are redundant.

a) Suppose $c = (\bar{l}' \vee \bar{b}_1 \vee \cdots \vee \bar{b}_r)$ is a newly learned clause. Prove that if $\bar{b}_j \in c$ is redundant, some other $\bar{b}_i \in c$ became false on the same level of the trail as $\bar{b}_j$ did.

b) Devise an efficient algorithm that discovers all of the redundant literals $\bar{b}_i$ in a given newly learned clause $c = (\bar{l}' \vee \bar{b}_1 \vee \cdots \vee \bar{b}_r)$. *Hint:* Use stamps.

258. [*21*] A non-decision literal l in Algorithm C's trail always has a reason $R_l = (l_0 \vee l_1 \vee \cdots \vee l_{k-1})$, where $l_0 = l$ and $\bar{l}_1, \ldots, \bar{l}_{k-1}$ precede l in the trail. Furthermore, the algorithm discovered this clause while looking at the watch list of l_1. True or false: $\bar{l}_2, \ldots, \bar{l}_{k-1}$ precede $\bar{l}_1$ in the trail. *Hint:* Consider Table 3 and its sequel.

259. [*M20*] Can `ACT(`j`)` exceed `ACT(`k`)` for values of ρ near 0 or 1, but not for *all* ρ?

260. [*18*] Describe in detail step C1's setting-up of `MEM`, the watch lists, and the trail.

261. [*21*] The main loop of Algorithm C is the unit-propagation process of steps C3 and C4. Describe the low-level details of link adjustment, etc., to be done in those steps.

262. [*20*] What low-level operations underlie changes to the heap in steps C6–C8?

263. [*21*] Write out the gory details by which step C7 constructs a new clause and step C9 puts it into the data structures of Algorithm C.

264. [*20*] Suggest a way by which Algorithm C could indicate progress by displaying "move codes" analogous to those of Algorithms A, B, D, and L. (See exercise 142.)

265. [*21*] Describe several circumstances in which the watched literals l_0 and/or l_1 of a clause c actually become false during the execution of Algorithm C.

266. [*20*] In order to keep from getting into a rut, CDCL solvers are often designed to make decisions at random, with a small probability p (say $p = .02$), instead of always choosing a variable of maximum activity. How would this policy change step C6?

▶ **267.** [*25*] Instances of SAT often contain numerous binary clauses, which are handled efficiently by the unit-propagation loop (62) of Algorithm L but not by the corresponding loop in step C3 of Algorithm C. (The technique of watched literals is great for long

anti-maximal-element
variable elimination
subsumption
CDCL solver
Sörensson
redundant
stamps
activity scores
damping factor
unit-propagation
move codes
watched literals
random decisions
binary clauses
unit-propagation
watched literals

clauses, but it is comparatively cumbersome for short ones.) What additional data structures will speed up Algorithm C's inner loop, when binary clauses are abundant?

268. [*21*] When Algorithm C makes a literal false at level 0 of the trail, we can remove it from all of the clauses. Such updating might take a long time, if we did it "eagerly"; but there's a lazy way out: We can delete a permanently false literal if we happen to encounter it in step C3 while looking for a new literal to watch (see exercise 261).

Explain how to adapt the `MEM` data structure conventions so that such deletions can be done *in situ*, without copying clauses from one location into another.

269. [*23*] Suppose Algorithm C reaches a conflict at level d of the trail, after having chosen the decision literals $u_1, u_2, \ldots, u_d$. Then the "trivial clause" $(\bar{l}' \vee \bar{u}_1 \vee \cdots \vee \bar{u}_{d'})$ must be true if the given clauses are satisfiable, where l' and d' are defined in step C7.

a) Show that, if we start with the clause $(\bar{l}' \vee \bar{b}_1 \vee \cdots \vee \bar{b}_r)$ that is obtained in step C7 and then resolve it somehow with zero or more known clauses, we can always reach a clause that subsumes the trivial clause.

b) Sometimes, as in (115), the clause that is slated to be learned in step C9 is much longer than the trivial clause. Construct an example in which $d = 3$, $d' = 1$, and $r = 10$, yet none of $\bar{b}_1, \ldots, \bar{b}_r$ are redundant in the sense of exercise 257.

c) Suggest a way to improve Algorithm C accordingly.

270. [*25*] (*On-the-fly subsumption.*) The intermediate clauses that arise in step C7, immediately after resolving with a reason R_l, occasionally turn out to be equal to the shorter clause $R_l \setminus l$. In such cases we have an opportunity to *strengthen* that clause by deleting l from it, thus making it potentially more useful in the future.

a) Construct an example where two clauses can each be subsumed in this way while resolving a single conflict. The subsumed clauses should both contain two literals assigned at the current level in the trail, as well as one literal from a lower level.

b) Show that it's easy to recognize such opportunities, and to strengthen such clauses efficiently, by modifying the steps of answer 263.

▸ **271.** [*25*] The sequence of learned clauses $C_1, C_2, \ldots$ often includes cases where C_i subsumes its immediate predecessor, C_{i-1}. In such cases we might as well *discard* C_{i-1}, which appears at the very end of `MEM`, and store C_i in its place, unless C_{i-1} is still in use as a reason for some literal on the trail. (For example, more than 8,600 of the 52,000 clauses typically learned from *waerden*$(3, 10; 97)$ by Algorithm C can be discarded in this way. Such discards are different from the on-the-fly subsumptions considered in exercise 270, because the subsumed C_{i-1} includes only one literal from its original conflict level; furthermore, learned clauses have usually been significantly simplified by the procedure of exercise 257, unless they're trivial.)

Design an efficient way to discover when C_{i-1} can be safely discarded.

272. [*30*] Experiment with the following idea: The clauses of *waerden*$(j, k; n)$ are symmetrical under reflection, in the sense that they remain unchanged overall if we replace x_k by $x_k^R = x_{n+1-k}$ for $1 \le k \le n$. Therefore, whenever Algorithm C learns a clause $C = (\bar{l}' \vee \bar{b}_1 \vee \cdots \vee \bar{b}_r)$, it is also entitled to learn the reflected clause $C^R = (\bar{l}'^R \vee \bar{b}_1^R \vee \cdots \vee \bar{b}_r^R)$.

273. [*27*] A clause C that is learned from *waerden*$(j, k; n)$ is valid also with respect to *waerden*$(j, k; n')$ when $n' > n$; and so are the clauses $C + i$ that are obtained by adding i to each literal of C, for $1 \le i \le n' - n$. For example, the fact that '35' follows from *waerden*$(3, 3; 7)$ allows us to add the clauses 35, 46, 57 to *waerden*$(3, 3; 9)$.

a) Exploit this idea to speed up the calculation of van der Waerden numbers.

level 0
eagerly
lazy
MEM
in situ
trivial clause
redundant
On-the-fly subsumption
subsumption
strengthen
learned clauses, sequence of
subsumes
discard
MEM
waerden
on-the-fly subsumptions
waerden
symmetrical
reflection
van der Waerden numbers

b) Explain how to apply it also to bounded model checking.

274. [*35*] Algorithm C sets the "reason" for a literal l as soon as it notices a clause that forces l to be true. Later on, other clauses that force l are often encountered, in practice; but Algorithm C ignores them, even though one of them might be a "better reason." (For example, another forcing clause might be significantly shorter.) Explore a modification of Algorithm C that tries to improve the reasons of non-decision literals.

▸ **275.** [*22*] Adapt Algorithm C to the problem of finding the lexicographically smallest solution to a satisfiability problem, by incorporating the ideas of exercise 109.

276. [*M15*] True or false: If F is a family of clauses and L is a set of strictly distinct literals, then $F \wedge L \vdash_1 \epsilon$ if and only if $(F \mid L) \vdash_1 \epsilon$.

277. [*M18*] If $(C_1, \ldots, C_t)$ is a certificate of unsatisfiability for F, and if all clauses of F have length ≥ 2, prove that some C_i is a unit clause.

278. [*22*] Find a six-step certificate of unsatisfiability for *waerden*$(3, 3; 9)$.

279. [*M20*] True or false: Every unsatisfiable 2SAT problem has a certificate '(l, ϵ)'.

▸ **280.** [*M26*] The problem *cook*(j, k) consists of all $\binom{n}{j}$ positive j-clauses and all $\binom{n}{k}$ negative k-clauses on $\{1, \ldots, n\}$, where $n = j + k - 1$. For example, *cook*$(2, 3)$ is

$$\{12,\ 13,\ 14,\ 23,\ 24,\ 34,\ \bar{1}\bar{2}\bar{3},\ \bar{1}\bar{2}\bar{4},\ \bar{1}\bar{3}\bar{4},\ \bar{2}\bar{3}\bar{4}\}.$$

a) Why are these clauses obviously unsatisfiable?
b) Find a totally positive certificate for *cook*(j, k), of length $\binom{n-1}{j-1}$.
c) Prove in fact that Algorithm C always learns *exactly* $\binom{n-1}{j-1}$ clauses when it proves the unsatisfiability of *cook*(j, k), if $M_{\rm p} = M_{\rm f} = \infty$ (no purging or flushing).

281. [*21*] Construct a certificate of unsatisfiability that refutes (99), (100), (101).

▸ **282.** [*M33*] Construct a certificate of unsatisfiability for the clauses *fsnark*(q) of exercise 176 when $q \geq 3$ is odd, using $O(q)$ clauses, all having length ≤ 4. *Hint:* Include the clauses $(\bar{a}_{j,p} \vee \bar{e}_{j,p})$, $(\bar{a}_{j,p} \vee \bar{f}_{j,p})$, $(\bar{e}_{j,p} \vee \bar{f}_{j,p})$, and $(a_{j,p} \vee e_{j,p} \vee f_{j,p})$ for $1 \leq j \leq q$, $1 \leq p \leq 3$.

283. [*HM46*] Does Algorithm C solve the flower snark problem in linear time? More precisely, let $p_q(M)$ be the probability that the algorithm refutes *fsnark*(q) while making at most M references to `MEM`. Is there a constant N such that $p_q(Nq) > \frac{1}{2}$ for all q?

284. [*23*] Given F and $(C_1, \ldots, C_t)$, a certificate-checking program tests condition (119) by verifying that F and clauses $C_1, \ldots, C_{i-1}$ will force a conflict when they are augmented by the unit literals of $\overline{C}_i$. While doing this, it can mark each clause of $F \cup \{C_1, \ldots, C_{i-1}\}$ that was reduced to a unit during the forcing process; then the truth of C_i does not depend on the truth of any unmarked clause.

In practice, many clauses of F are never marked at all, hence F will remain unsatisfiable even if we leave them out. Furthermore, many clauses C_i are not marked during the verification of any of their successors, $\{C_{i+1}, \ldots, C_t\}$; such clauses C_i needn't be verified, nor need we mark any of the clauses on which they depend.

Therefore we can save work by checking the certificate backwards: Start by marking the final clause C_t, which is ϵ and always needs to be verified. Then, for $i = t, t-1, \ldots,$ check C_i only if it has been marked.

The unit propagations can all be done without recording the "reason" R_l that has caused any literal l to be forced. In practice, however, many of the forced literals don't actually contribute to the conflicts that arise, and we don't want to mark any clauses that aren't really involved.

Explain how to use reasons, as in Algorithm C, so that clauses are marked by the verifier only if they actually participate in the proof of a marked clause C_i.

bounded model checking
better reason
lexicographically smallest solution
notation $F \mid L$
notation $F \vdash_1 \epsilon$
certificate of unsatisfiability
unit clause
2SAT
positive j-clauses
negative k-clauses
cook
purging
flushing
maximal elements
fsnark
flower snark
reason

285. [*19*] Using the data in Fig. 50, the text observes that Eq. (124) gives $j = 95$, $s_j = 3081$, and $m_j = 59$ when $\alpha = \frac{15}{16}$. What are j, s_j, and m_j when (a) $\alpha = \frac{9}{16}$? (b) $\alpha = \frac{1}{2}$? (c) $\alpha = \frac{7}{16}$? Also compare the effectiveness of different α's by computing the number b_j of "black" clauses (those with $0 < \texttt{RANGE}(c) < j$ that proved to be useful).

286. [*M24*] What choice of signatures-to-keep in Fig. 50 is *optimum*, in the sense that it maximizes $\sum b_{pq}x_{pq}$ subject to the conditions $\sum a_{pq}x_{pq} \le 3114$, $x_{pq} \in \{0,1\}$, and $x_{pq} \ge x_{p'q'}$ for $1 \le p \le p' \le 7$, $0 \le q \le q' \le 8$? Here a_{pq} and b_{pq} are the areas of the gray and black clauses that have signature (p,q), as given by the matrices in the text. [This is a special case of the "knapsack problem with a partial ordering."]

287. [*25*] What changes to Algorithm C are necessary to make it do a "full run," and later to learn from all of the conflicts that arose during that run?

288. [*28*] Spell out the details of computing `RANGE` scores and then compressing the database of learned clauses, during a round of purging.

289. [*M20*] Assume that the kth round of purging begins with y_k clauses in memory after $k\Delta + \binom{k}{2}\delta$ clauses have been learned, and that purging removes $\frac{1}{2}y_k$ of those clauses. Find a closed formula for y_k as a function of k.

290. [*17*] Explain how to find x_k, the unassigned variable of maximum activity that is used for flushing literals. *Hint:* It's in the `HEAP` array.

291. [*20*] In the text's hypothetical scenario about flushing Table 3 back to level 15, why will 49 soon appear on the trail, instead of $\overline{49}$?

292. [*M21*] How large can `AGILITY` get after repeatedly executing (127)?

293. [*21*] Spell out the details of updating $M_{\rm f}$ to $M+\Delta_{\rm f}$ when deciding whether or not to flush. Also compute the agility threshold that's specified in Table 4. *Hint:* See (131).

294. [*HM21*] For each binary vector $\alpha = x_1x_2x_3x_4$, find the generating function $g_\alpha(z) = \sum_{j=0}^{\infty} p_{\alpha,j}z^j$, where $p_{\alpha,j}$ is the probability that Algorithm P will solve the seven clauses of (7) after making exactly j flips, given the initial values α in step P1. Deduce the mean and variance of the number of steps needed to find a solution.

295. [*M23*] Algorithm P often finds solutions much more quickly than predicted by Corollary W. But show that some 3SAT clauses will indeed require $\Omega((4/3)^n)$ trials.

296. [*HM20*] Complete the proof of Theorem U by (approximately) maximizing the quantity $f(p,q)$ in (129). *Hint:* Consider $f(p+1,q)/f(p,q)$.

▸ **297.** [*HM26*] (Emo Welzl.) Let $G_q(z) = \sum_p C_{p,p+q-1}(z/3)^{p+q}(2z/3)^p$ be the generating function for stopping time $t = 2p+q$ when $Y_0 = q$ in the proof of Theorem U.

a) Find a closed form for $G_q(z)$, using formulas from Section 7.2.1.6.
b) Explain why $G_q(1)$ is less than 1.
c) Evaluate and interpret the quantity $G_q'(1)/G_q(1)$.
d) Use Markov's inequality to bound the probability that $Y_t = 0$ for some $t \le N$.
e) Show that Corollary W follows from this analysis.

298. [*HM22*] Generalize Theorem U and Corollary W to the case where each clause has at most k literals, where $k \ge 3$.

299. [*HM23*] Continuing the previous exercise, investigate the case $k = 2$.

▸ **300.** [*25*] Modify Algorithm P so that it can be implemented with bitwise operations, thereby running (say) 64 independent trials simultaneously.

▸ **301.** [*25*] Discuss implementing the algorithm of exercise 300 efficiently on `MMIX`.

signature
knapsack problem with a partial ordering
full run
compressing
purging
flushing literals
`HEAP`
`AGILITY`
agility threshold
generating function
variance
Welzl
Markov's inequality
bitwise operations
broadword computation
`MMIX`

302. [*26*] Expand the text's high-level description of steps W4 and W5, by providing low-level details about exactly what the computer should do.

303. [*HM20*] Solve exercise 294 with Algorithm W in place of Algorithm P.

304. [*HM34*] Consider the 2SAT problem with $n(n-1)$ clauses $(\bar{x}_j \vee x_k)$ for all $j \neq k$. Find the generating functions for the number of flips taken by Algorithms P and W. *Hint:* Exercises 1.2.6–68 and MPR–105 are helpful for finding the exact formulas.

▶ **305.** [*M25*] Add one more clause, $(\bar{x}_1 \vee \bar{x}_2)$, to the previous exercise and find the resulting generating functions when $n = 4$. What happens when $p = 0$ in Algorithm W?

▶ **306.** [*HM32*] (Luby, Sinclair, and Zuckerman, 1993.) Consider a "Las Vegas algorithm" that succeeds or fails; it succeeds at step t with probability p_t, and fails with probability $p_\infty < 1$. Let $q_t = p_1 + p_2 + \cdots + p_t$ and $E_t = p_1 + 2p_2 + \cdots + tp_t$; also let $E_\infty = \infty$ if $p_\infty > 0$, otherwise $E_\infty = \sum_t tp_t$. (The latter sum might be ∞.)

a) Suppose we abort the algorithm and restart it again, whenever the first N steps have not succeeded. Show that if $q_N > 0$, this strategy will succeed after performing an average of $l(N) < \infty$ steps. What is $l(N)$?
b) Compute $l(N)$ when $p_m = \frac{m}{n}$, $p_\infty = \frac{n-m}{n}$, otherwise $p_t = 0$, where $1 \leq m \leq n$.
c) Given the uniform distribution, $p_t = \frac{1}{n}$ for $1 \leq t \leq n$, what is $l(N)$?
d) Find all probability distributions such that $l(N) = l(1)$ for all $N \geq 1$.
e) Find all probability distributions such that $l(N) = l(n)$ for all $N \geq n$.
f) Find all probability distributions such that $q_{n+1} = 1$ and $l(n) \leq l(n+1)$.
g) Find all probability distributions such that $q_3 = 1$ and $l(1) < l(3) < l(2)$.
h) Let $l = \inf_{N\geq 1} l(N)$, and let N^* be the least positive integer such that $l(N^*) = l$, or ∞ if no such integer exists. Prove that $N^* = \infty$ implies $l = E_\infty < \infty$.
i) Find N^* for the probability distribution $p_t = [t > n]/((t-n)(t+1-n))$, given $n \geq 0$.
j) Exhibit a simple example of a probability distribution for which $N^* = \infty$.
k) Let $L = \min_{t\geq 1} t/q_t$. Prove that $l \leq L \leq 2l - 1$.

307. [*HM28*] Continuing exercise 306, consider a more general strategy defined by an infinite sequence of positive integers $(N_1, N_2, \ldots)$: "Set $j \leftarrow 0$; then, while success has not yet been achieved, set $j \leftarrow j+1$ and run the algorithm with cutoff parameter N_j."

a) Explain how to compute $\mathrm{E}\,X$, where X is the number of steps taken before this strategy succeeds.
b) Let $T_j = N_1 + \cdots + N_j$. Prove that $\mathrm{E}\,X = \sum_{j=1}^{\infty} \Pr(T_{j-1} < X \leq T_j)\, l(N_j)$, if we have $q_{N_j} > 0$ for all j.
c) Consequently the steady strategy $(N^*, N^*, \ldots)$ is best: $\mathrm{E}\,X \geq l(N^*) = l$.
d) Given n, exercise 306(b) defines n simple probability distributions $p^{(m)}$ that have $l(N^*) = n$, but the value of $N^* = m$ is different in each case. Prove that any sequence $(N_1, N_2, \ldots)$ must have $\mathrm{E}\,X > \frac{1}{4}nH_n - \frac{1}{2}n = \frac{1}{4}lH_l - \frac{1}{2}l$ on at least one of those $p^{(m)}$. *Hint:* Consider the smallest r such that, for each m, the probability is $\geq \frac{1}{2}$ that r trial runs suffice; show that $\geq n/(2m)$ of $\{N_1, \ldots, N_r\}$ are $\geq m$.

308. [*M29*] This exercise explores the "reluctant doubling" sequence (130).

a) What is the smallest n such that $S_n = 2^a$, given $a \geq 0$?
b) Show that $\{n \mid S_n = 1\} = \{2k + 1 - \nu k \mid k \geq 0\}$; hence the generating function $\sum_n z^n [S_n = 1]$ is the infinite product $z(1+z)(1+z^3)(1+z^7)(1+z^{15}) \ldots$.
c) Find similar expressions for $\{n \mid S_n = 2^a\}$ and $\sum_n z^n [S_n = 2^a]$.
d) Let $\Sigma(a,b,k) = \sum_{n=1}^{r(a,b,k)} S_n$, where $S_{r(a,b,k)}$ is the $2^b k$th occurrence of 2^a in $\langle S_n \rangle$. For example, $\Sigma(1,0,3) = S_1 + \cdots + S_{10} = 16$. Evaluate $\Sigma(a,b,1)$ in closed form.
e) Show that $\Sigma(a,b,k{+}1) - \Sigma(a,b,k) \leq (a + b + 2k - 1)2^{a+b}$, for all $k \geq 1$.

WalkSAT+
analysis of algorithms
Luby
Sinclair
Zuckerman
Las Vegas algorithm
uniform distribution
reluctant doubling
generating function

f) Given any probability distribution as in exercise 306(k), let $a = \lceil \lg t \rceil$ and $b = \lceil \lg 1/q_t \rceil$, where $t/q_t = L$; thus $L \le 2^{a+b} < 4L$. Prove that if the strategy of exercise 307 is used with $N_j = S_j$, we have

$$\mathrm{E}\,X \le \Sigma(a,b,1) + \sum_{k\ge1} Q^k\bigl(\Sigma(a,b,k{+}1) - \Sigma(a,b,k)\bigr), \quad \text{where } Q = (1-q_{2^a})^{2^b}.$$

g) Therefore $\langle S_n \rangle$ gives $\mathrm{E}\,X < 13l \lg l + 49l$, for every probability distribution.

309. [*20*] Exercise 293 explains how to use the reluctant doubling sequence with Algorithm C. Is Algorithm C a Las Vegas algorithm?

310. [*M25*] Explain how to compute the "reluctant Fibonacci sequence"

$$1, 1, 2, 1, 2, 3, 1, 1, 2, 3, 5, 1, 1, 2, 1, 2, 3, 5, 8, 1, 1, 2, 1, 2, 3, 1, 1, 2, 3, 5, 8, 13, 1, \ldots,$$

which is somewhat like (130) and useful as in exercise 308, but its elements are Fibonacci numbers instead of powers of 2.

311. [*21*] Compute approximate values of $\mathrm{E}\,X$ for the 100 probability distributions of exercise 306(b) when $n = l = 100$, using the method of exercise 307 with the sequences $\langle S_n \rangle$ of exercise 308 and $\langle S'_n \rangle$ of exercise 310. Also consider the more easily generated "ruler doubling" sequence $\langle R_n \rangle$, where $R_n = n \mathbin{\&} -n = 2^{\rho n}$. Which sequence is best?

312. [*HM24*] Let $T(m,n) = \mathrm{E}\,X$ when the reluctant doubling method is applied to the probability distribution defined in exercise 306(b). Express $T(m,n)$ in terms of the generating functions in exercise 308(c).

▸ **313.** [*22*] Algorithm W always flips a cost-free literal if one is present in C_j, without considering its parameter p. Show that such a flip always decreases the number of unsatisfied clauses, r; but it might *increase* the distance from x to the nearest solution.

▸ **314.** [*36*] (H. H. Hoos, 1998.) If the given clauses are satisfiable, and if $p > 0$, can there be an initial x for which Algorithm W always loops forever?

315. [*M18*] What value of p is appropriate in Theorem J when $d = 1$?

316. [*HM20*] Is Theorem J a consequence of Theorem L?

▸ **317.** [*M26*] Let $\alpha(G) = \Pr(\overline{A}_1 \cap \cdots \cap \overline{A}_m)$ under the assumptions of (133), when $p_i = p = (d-1)^{d-1}/d^d$ for $1 \le i \le m$ and every vertex of G has degree at most $d > 1$. Prove, by induction on m, that $\alpha(G) > 0$ and that $\alpha(G) > \frac{d-1}{d}\alpha(G \setminus v)$ when v has degree $< d$.

318. [*HM27*] (J. B. Shearer.) Prove that Theorem J is the best possible result of its kind: If $p > (d-1)^{d-1}/d^d$ and $d > 1$, there is a graph G of maximum degree d for which $(p, \ldots, p) \notin \mathcal{R}(G)$. *Hint:* Consider complete t-ary trees, where $t = d - 1$.

319. [*HM20*] Show that $pde < 1$ implies $p \le (d-1)^{d-1}/d^d$.

320. [*M24*] Given a lopsidependency graph G, the *occurrence threshold* $\rho(G)$ is the smallest value p such that it's sometimes impossible to avoid all events when each event occurs with probability p. For example, the Möbius polynomial for the path P_3 is $1-p_1-p_2-p_3+p_1p_3$; so the occurrence threshold is ϕ^{-2}, the least p with $1-3p+p^2 \le 0$.

a) Prove that the occurrence threshold for P_m is $1/(4\cos^2\frac{\pi}{m+2})$.

b) What is the occurrence threshold for the cycle graph C_m?

321. [*M24*] Suppose each of four random events A, B, C, D occurs with probability p, where $\{A, C\}$ and $\{B, D\}$ are independent. According to exercise 320(b) with $m = 4$, there's a joint distribution of (A, B, C, D) such that at least one of the events always occurs, whenever $p \ge (2-\sqrt{2})/2 \approx 0.293$. Exhibit such a distribution when $p = 3/10$.

Las Vegas
reluctant Fibonacci sequence
Fibonacci numbers
ruler doubling
Hoos
Shearer
complete t-ary trees
lopsidependency graph
occurrence threshold
Möbius polynomial
path
phi
cycle graph

▶ **322.** [*HM35*] (K. Kolipaka and M. Szegedy, 2011.) Surprisingly, the previous exercise *cannot* be solved in the setting of Algorithm M! Suppose we have independent random variables (W, X, Y, Z) such that A depends on W and X, B depends on X and Y, C depends on Y and Z, D depends on Z and W. Here W equals j with probability w_j for all integers j; X, Y, and Z are similar. This exercise will prove that the constraint $\overline{A}\cap\overline{B}\cap\overline{C}\cap\overline{D}$ is always satisfiable, even when p is as large as 0.333.

a) Express the probability $\Pr(\overline{A}\cap\overline{B}\cap\overline{C}\cap\overline{D})$ in a convenient way.

b) Suppose there's a distribution of W, X, Y, Z with $\Pr(A)=\Pr(B)=\Pr(C)=\Pr(D)=p$ and $\Pr(\overline{A}\cap\overline{B}\cap\overline{C}\cap\overline{D})=0$. Show that there are ten values such that

$$\begin{aligned} 0\le a,b,c,d,a',b',c',d'\le 1, &\qquad 0<\mu,\nu<1,\\ \mu a+(1-\mu)a'\le p, &\qquad \mu b+(1-\mu)b'\le p,\\ \nu c+(1-\nu)c'\le p, &\qquad \nu d+(1-\nu)d'\le p,\\ a+d\ge 1 \text{ or } b+c\ge 1, &\qquad a+d'\ge 1 \text{ or } b+c'\ge 1,\\ a'+d\ge 1 \text{ or } b'+c\ge 1, &\qquad a'+d'\ge 1 \text{ or } b'+c'\ge 1. \end{aligned}$$

c) Find all solutions to those constraints when $p=1/3$.

d) Convert those solutions to distributions that have $\Pr(\overline{A}\cap\overline{B}\cap\overline{C}\cap\overline{D})=0$.

323. [*10*] What trace precedes *ccb* in the list (135)?

▶ **324.** [*22*] Given a trace $\alpha = x_1x_2\ldots x_n$ for a graph G, explain how to find all strings β that are equivalent to α, using Algorithm 7.2.1.2V. How many strings yield (136)?

▶ **325.** [*20*] An *acyclic orientation* of a graph G is an assignment of directions to each of its edges so that the resulting digraph has no oriented cycles. Show that the number of traces for G that are *permutations* of the vertices (with each vertex appearing exactly once in the trace) is the number of acyclic orientations of G.

326. [*20*] True or false: If α and β are traces with $\alpha=\beta$, then $\alpha^R=\beta^R$. (See (137).)

▶ **327.** [*22*] Design an algorithm to multiply two traces α and β, when clashing is defined by territory sets $T(a)$ in some universe U. Assume that U is small (say $|U|\le 64$), so that bitwise operations can be used to represent the territories.

328. [*20*] Continuing exercise 327, design an algorithm that computes α/β. More precisely, if β is a right factor of α, in the sense that $\alpha=\gamma\beta$ for some trace γ, your algorithm should compute γ; otherwise it should report that β is *not* a right factor.

329. [*21*] Similarly, design an algorithm that either computes $\alpha\setminus\beta$ or reports that α isn't a left factor of β.

▶ **330.** [*21*] Given any graph G, explain how to define territory sets $T(a)$ for its vertices a in such a way that we have $a=b$ or $a \mathrel{-\!\!-} b$ if and only if $T(a)\cap T(b)\ne\emptyset$. (Thus traces can always be modeled by empilements of pieces.) Under what circumstances is it possible to do this with $|T(a)|=2$ for all a, as in the text's example (136)?

331. [*M20*] What happens if the right-hand side of (139) is expanded without allowing *any* of the variables to commute with each other?

332. [*20*] When a trace is represented by its lexicographically smallest string, no letter in that representative string is followed by a smaller letter with which it commutes. (For example, no c is followed by a in (135), because we could get an equivalent smaller string by changing ca to ac.)

Conversely, given any ordered set of letters, some of which commute, consider all strings having no letter followed by a smaller letter with which it commutes. Is every such string the lexicographically smallest of its trace?

Kolipaka
Szegedy
acyclic orientation
digraph
territory sets
bitwise operations
territories
multiplication of traces
right factor
right division of traces
left factor
left division of traces
territory sets
intersection graphs
empilements

▶ **333.** [*M20*] (Carlitz, Scoville, and Vaughan, 1976.) Let D be a digraph on $\{1,\ldots,m\}$, and let A be the set of all strings $a_{j_1}\ldots a_{j_n}$ such that $j_i \longrightarrow j_{i+1}$ in D for $1 \le i < n$. Similarly let B be the set of all strings $a_{j_1}\ldots a_{j_n}$ such that $j_i \not\longrightarrow j_{i+1}$ for $1 \le i < n$. Prove that

$$\sum_{\alpha\in A}\alpha \;=\; 1\Big/\sum_{\beta\in B}(-1)^{|\beta|}\beta \;=\; \sum_{k\ge 0}\Bigl(1-\sum_{\beta\in B}(-1)^{|\beta|}\beta\Bigr)^k$$

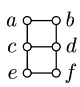
is an identity in the noncommutative variables $\{a_1,\ldots,a_m\}$. (For example, we have

$$1+a+b+ab+ba+aba+bab+\cdots \;=\; \sum_{k\ge 0}(a+b-aa-bb+aaa+bbb-\cdots)^k$$

in the case $m=2$, $1\not\longrightarrow 1$, $1\longrightarrow 2$, $2\longrightarrow 1$, $2\not\longrightarrow 2$.)

▶ **334.** [*25*] Design an algorithm to generate all traces of length n that correspond to a given graph on the alphabet $\{1,\ldots,m\}$, representing each trace by its lexicographically smallest string.

335. [*HM26*] If the vertices of G can be ordered in such a way that $x<y<z$ and $x \not- y$ and $y \not- z$ implies $x \not- z$, show that the Möbius series M_G can be expressed as a determinant. For example,

if $G =$ 

then

$$M_G = \det\begin{pmatrix} 1-a & -b & -c & 0 & 0 & 0\\ -a & 1-b & 0 & -d & 0 & 0\\ -a & -b & 1-c & -d & -e & 0\\ -a & -b & -c & 1-d & 0 & -f\\ -a & -b & -c & -d & 1-e & -f\\ -a & -b & -c & -d & -e & 1-f\end{pmatrix}.$$

▶ **336.** [*M20*] If graphs G and H on distinct vertices have the Möbius series M_G and M_H, what are the Möbius series for (a) $G\oplus H$ and (b) $G \mathbin{-\!\!-} H$?

337. [*M20*] Suppose we obtain the graph G' from G by substituting a clique of vertices $\{a_1,\ldots,a_k\}$ for some vertex a, then including edges from a_j to each neighbor of a for $1\le j\le k$. Describe the relation between $M_{G'}$ and M_G.

338. [*M21*] Prove Viennot's general identity (144) for source-constrained traces.

▶ **339.** [*HM26*] (G. Viennot.) This exercise explores factorization of traces into pyramids.

a) Each letter x_j of a given trace $\alpha = x_1\ldots x_n$ lies at the top of a unique pyramid β_j such that β_j is a left factor of α. For example, in the trace *bcebafdc* of (136), the pyramids $\beta_1,\ldots,\beta_8$ are respectively *b*, *bc*, *e*, *bcb*, *bcba*, *ef*, *bced*, and *bcebdc*. Explain intuitively how to find these pyramidal left factors from α's empilement.

b) A *labeled trace* is an assignment of distinct numbers to the letters of a trace; for example, *abca* might become $a_4b_7c_6a_3$. A *labeled pyramid* is the special case when the pyramid's top element is required to have the smallest label. Prove that every labeled trace is uniquely factorizable into labeled pyramids whose topmost labels are in ascending order. (For example, $b_6c_2e_4b_7a_8f_5d_1c_3 = b_6c_2e_4d_1\cdot b_7a_8c_3\cdot f_5$.)

c) Suppose there are t_n traces of length n, and p_n pyramids. Then there are $T_n = n!\,t_n$ labeled traces and $P_n=(n-1)!\,p_n$ labeled pyramids (because only the relative order of the labels is significant). Letting $T(z)=\sum_{n\ge 0}T_nz^n/n!$ and $P(z)=\sum_{n\ge 1}P_nz^n/n!$, prove that the number of labeled traces of length n whose factorization in part (b) has exactly l pyramids is $n!\,[z^n]\,P(z)^l/l!$.

d) Consequently $T(z)=e^{P(z)}$.

e) Therefore (and this is the punch line!) $\ln M_G(z) = -\sum_{n\ge 1}p_nz^n/n$.

Carlitz
Scoville
Vaughan
digraph
noncommutative variables
lexicographically smallest
determinant
Möbius series
direct sum of graphs
join of graphs
clique
Viennot
Viennot
pyramids
left factor
labeled trace
labeled pyramid
generating functions, exponential

▶ **340.** [*M20*] If we assign a weight $w(\sigma)$ to every cyclic permutation σ, then every permutation π has a weight $w(\pi)$ that is the product of the weights of its cycles. For example, if $\pi = \binom{1\,2\,3\,4\,5\,6\,7}{3\,1\,4\,2\,7\,6\,5} = (1\,3\,4\,2)(5\,7)(6)$ then $w(\pi) = w((1\,3\,4\,2))w((5\,7))w((6))$.

The *permutation polynomial* of a set S is the sum of $w(\pi)$ over all permutations of S. Given any $n \times n$ matrix $A = (a_{ij})$, show that it's possible to define appropriate cycle weights so that the permutation polynomial of $\{1, \ldots, n\}$ is the determinant of A.

341. [*M25*] The *involution polynomial* of a set S is the special case of the permutation polynomial when the cycle weights have the form $w_{jj}x$ for the 1-cycle (j) and $-w_{ij}$ for the 2-cycle $(i\,j)$, otherwise $w(\sigma) = 0$. For example, the involution polynomial of $\{1, 2, 3, 4\}$ is $w_{11}w_{22}w_{33}w_{44}x^4 - w_{11}w_{22}w_{34}x^2 - w_{11}w_{23}w_{44}x^2 - w_{11}w_{24}w_{33}x^2 - w_{12}w_{33}w_{44}x^2 - w_{13}w_{22}w_{44}x^2 - w_{14}w_{22}w_{33}x^2 + w_{12}w_{34} + w_{13}w_{24} + w_{14}w_{23}$.

Prove that, if $w_{ij} > 0$ for $1 \le i \le j \le n$, the involution polynomial of $\{1, \ldots, n\}$ has n distinct real roots. *Hint:* Show also that, if the roots for $\{1, \ldots, n-1\}$ are $q_1 < \cdots < q_{n-1}$, then the roots r_k for $\{1, \ldots, n\}$ satisfy $r_1 < q_1 < r_2 < \cdots < q_{n-1} < r_n$.

342. [*HM25*] (Cartier and Foata, 1969.) Let G_n be the graph whose vertices are the $\sum_{k=1}^{n} \binom{n}{k}(k-1)!$ cyclic permutations of subsets of $\{1, \ldots, n\}$, with $\sigma \mathrel{-\!\!-} \tau$ when σ and τ intersect. For example, the vertices of G_3 are (1), (2), (3), (12), (13), (23), (123), (132); and they're mutually adjacent except that $(1) \not\mathrel{-\!\!-} (2)$, $(1) \not\mathrel{-\!\!-} (3)$, $(1) \not\mathrel{-\!\!-} (23)$, $(2) \not\mathrel{-\!\!-} (3)$, $(2) \not\mathrel{-\!\!-} (13)$, $(12) \not\mathrel{-\!\!-} (3)$. Find a beautiful relation between M_{G_n} and the characteristic polynomial of an $n \times n$ matrix.

▶ **343.** [*M25*] If G is any cograph, show that $(p_1, \ldots, p_m) \in \mathcal{R}(G)$ if and only if we have $M_G(p_1, \ldots, p_m) > 0$. Exhibit a non-cograph for which the latter statement is *not* true.

344. [*M33*] Given a graph G as in Theorem S, let $B_1, \ldots, B_m$ have the joint probability distribution of exercise MPR–31, with $\pi_I = 0$ whenever I contains distinct vertices $\{i, j\}$ with $i \mathrel{-\!\!-} j$, otherwise $\pi_I = \prod_{i \in I} p_i$.

a) Show that this distribution is legal (see exercise MPR–32) if $(p_1, \ldots, p_m) \in \mathcal{R}(G)$.
b) Show that this "extreme distribution" also satisfies condition (147).
c) Let $\beta(G) = \Pr(\overline{B}_1 \cap \cdots \cap \overline{B}_m)$. If $J \subseteq \{1, \ldots, m\}$, express $\beta(G|J)$ in terms of M_G.
d) Defining $\alpha(G)$ as in exercise 317, with events A_j satisfying (133) and probabilities $(p_1, \ldots, p_m) \in \mathcal{R}(G)$, show that $\alpha(G \mid J) \ge \beta(G \mid J)$ for all $J \subseteq \{1, \ldots, m\}$.
e) If p_i satisfies (134), show that $\beta(G|J) \ge \prod_{j \in J}(1 - \theta_j)$.

345. [*M30*] Construct unavoidable events that satisfy (147) when $(p_1, \ldots, p_m) \notin \mathcal{R}(G)$.

▶ **346.** [*HM28*] Write (142) as $M_G = M_{G \setminus a}(1 - aK_{a,G})$ where $K_{a,G} = M_{G \setminus a^*}/M_{G \setminus a}$.

a) If $(p_1, \ldots, p_m) \in \mathcal{R}(G)$, prove that $K_{a,G}$ is monotonic in all of its parameters: It does not increase if any of $p_1, \ldots, p_m$ are decreased.
b) Exploit this fact to design an algorithm that computes $M_G(p_1, \ldots, p_m)$ and decides whether or not $(p_1, \ldots, p_m) \in \mathcal{R}(G)$, given a graph G and probabilities $(p_1, \ldots, p_m)$. Illustrate your algorithm on the graph $G = P_3 \,\square\, P_2$ of exercise 335.

▶ **347.** [*M28*] A graph is called *chordal* when it has no induced cycle C_k for $k > 3$. Equivalently (see Section 7.4.2), a graph is chordal if and only if its edges can be defined by territory sets $T(a)$ that induce connected subgraphs of some tree. For example, interval graphs and forests are chordal.

a) Say that a graph is *tree-ordered* if its vertices can be arranged as nodes of a forest in such a way that

$$\begin{aligned} &a \mathrel{-\!\!-} b \text{ implies } a \succ b \text{ or } b \succ a;\\ &a \succ b \succ c \text{ and } a \mathrel{-\!\!-} c \text{ implies } a \mathrel{-\!\!-} b. \end{aligned} \qquad (*)$$

cyclic permutation
weighted permutations
permutations, weighted
permutation polynomial
determinant
involution polynomial
real roots of polynomials
interlaced roots
Cartier
Foata
characteristic polynomial
cograph
extreme distribution
monotonic
chordal
territory sets
interval graphs
forests

(Here '$a \succ b$' means that a is a proper ancestor of b in the forest.) Prove that every tree-ordered graph is chordal.

b) Conversely, show that every chordal graph can be tree-ordered.

c) Show that the algorithm in the previous exercise becomes quite simple when it is applied to a tree-ordered graph, if a is eliminated before b whenever $a \succ b$.

d) Consequently Theorem L can be substantially strengthened when G is a chordal graph: *When G is tree-ordered by $\succ$, the probability vector $(p_1, \ldots, p_m)$ is in $\mathcal{R}(G)$ if and only if there are numbers $0 \le \theta_1, \ldots, \theta_m < 1$ such that*

$$p_i \;=\; \theta_i \prod_{i\,—\,j \text{ in } G,\; i \succ j} (1-\theta_j).$$

348. [*HM26*] (A. Pringsheim, 1894.) Show that any power series $f(z) = \sum_{n=0}^{\infty} a_n z^n$ with $a_n \ge 0$ and radius of convergence ρ, where $0 < \rho < \infty$, has a singularity at $z = \rho$.

▸ **349.** [*M24*] Analyze Algorithm M *exactly* in the two examples considered in the text (see (150)): For each binary vector $x = x_1 \ldots x_7$, compute the generating function $g_x(z) = \sum_t p_{x,t} z^t$, where $p_{x,t}$ is the probability that step M3 will be executed exactly t times after step M1 produces x. Assume that step M2 always chooses the smallest possible value of j. (Thus the 'Case 2' scenario in (150) will never occur.)

What are the mean and variance of the running times, in (i) Case 1? (ii) Case 2?

▸ **350.** [*HM26*] (W. Pegden.) Suppose Algorithm M is applied to the $m = n + 1$ events

$$A_j = x_j \quad \text{for } 1 \le j \le n; \qquad A_m = x_1 \vee \cdots \vee x_n.$$

Thus A_m is true whenever any of the other A_j is true; so we could implement step M2 by never setting $j \leftarrow m$. Alternatively, we could decide to set $j \leftarrow m$ whenever possible. Let $(N_{\rm i}, N_{\rm ii}, N_{\rm iii}, N_{\rm iv}, N_{\rm v})$ be the number of resamplings performed when parameter ξ_k of the algorithm is (i) $1/2$; (ii) $1/(2n)$; (iii) $1/2^n$; (iv) $1/(n+k)$; (v) $1/(n+k)^2$.

a) Find the asymptotic mean and variance of each N, if j is never equal to m.

b) Find the asymptotic mean and variance of each N, if j is never less than m.

c) Let G be the graph on $\{1, \ldots, n+1\}$ with edges $j \,—\, (n+1)$ for $1 \le j \le n$, and let $p_j = \Pr(A_j)$. For which of the five choices of ξ_k is $(p_1, \ldots, p_{n+1}) \in \mathcal{R}(G)$?

▸ **351.** [*25*] The Local Lemma can be applied to the satisfiability problem for m clauses on n variables if we let A_j be the event "C_j is not satisfied." The dependency graph G then has $i \,—\, j$ whenever two clauses C_i and C_j share at least one common variable. If, say, C_i is $(x_3 \vee \bar{x}_5 \vee x_6)$, then (133) holds whenever $p_i \ge (1-\xi_3)\xi_5(1-\xi_6)$, assuming that each x_k is true with probability ξ_k, independent of the other x's.

But if, say, C_j is $(\bar{x}_2 \vee x_3 \vee x_7)$, condition (133) remains true even if we don't stipulate that $i \,—\, j$. Variable x_3 appears in both clauses, yet the cases when C_j is satisfied are never bad news for C_i. We need to require that $i \,—\, j$ in condition (133) only when C_i and C_j are "resolvable" clauses, namely when some variable occurs positively in one and negatively in the other.

Extend this reasoning to the general setting of Algorithm M, where we have arbitrary events A_j that depend on variables Ξ_j: Define a lopsidependency graph G for which (133) holds even though we might have $i \not\!\!—\, j$ in some cases when $\Xi_i \cap \Xi_j \ne \emptyset$.

352. [*M21*] Show that $E_j \le \theta_j/(1-\theta_j)$ in (152), when (134) holds.

353. [*M21*] Consider Case 1 and Case 2 of Algorithm M as illustrated in (150).

a) How many solutions $x_1 \ldots x_n$ are possible? (Generalize from $n = 7$ to any n.)

b) How many solutions are predicted by Theorem S?

proper ancestor
tree-ordered graph
Pringsheim
nonnegative coefficients
analysis of algorithms
variance
Pegden
asymptotic
Local Lemma
dependency graph
resolvable
lopsidependency graph

c) Show that in Case 2 the lopsidependency graph is much smaller than the dependency graph. How many solutions are predicted when the smaller graph is used?

354. [*HM20*] Show that the expected number E N of resampling steps in Algorithm M is at most $-M_G^{*\prime}(1)/M_G^*(1)$.

355. [*HM21*] In (152), prove that $E_j \le 1/\delta$ when $(p_1,\ldots,p_m)$ has positive slack δ. *Hint:* Consider replacing p_j by $p_j + \delta p_j$.

▶ **356.** [*M33*] (*The Clique Local Lemma.*) Let G be a graph on $\{1,\ldots,m\}$, and let $G\mid U_1, \ldots, G\mid U_t$ be cliques that cover all the edges of G. Assign numbers $\theta_{ij} \ge 0$ to the vertices of each U_j, such that $\Sigma_j = \sum_{i\in U_j}\theta_{ij} < 1$. Assume that

$$\Pr(A_i) = p_i \le \theta_{ij} \prod_{k\ne j,\, i\in U_k} (1+\theta_{ik}-\Sigma_k) \quad \text{whenever } 1\le i\le m \text{ and } i\in U_j.$$

a) Prove that $(p_1,\ldots,p_m)\in\mathcal{R}(G)$. *Hint:* Letting $\overline{A}_S$ denote $\bigcap_{i\in S}\overline{A}_i$, show that

$$\Pr(A_i \mid \overline{A}_S) \;\le\; \theta_{ij} \quad \text{whenever } 1\le i\le m \text{ and } i\in U_j \text{ and } S\cap U_j=\emptyset.$$

b) Also E_i in (152) is at most $\min_{i \text{---} j \text{ in } G}\theta_{ij}/(1-\Sigma_j)$. (See Theorems M and K.)

c) Improve Theorem L by showing that, if $0\le\theta_j<\frac12$, then $(p_1,\ldots,p_m)\in\mathcal{R}(G)$ when

$$p_i \;=\; \theta_i\Bigl(\prod_{i \text{---} j \text{ in } G}(1-\theta_j)\Bigr)\Big/ \max_{i \text{---} j \text{ in } G}(1-\theta_j).$$

▶ **357.** [*M20*] Let $x=\pi_{\bar v}$ and $y=\pi_v$ in (155), and suppose the field of variable v is (p,q,r). Express x and y as functions of p, q, and r.

358. [*M20*] Continuing exercise 357, prove that $r=\max(p,q,r)$ if and only if $x,y\ge\frac12$.

359. [*20*] Equations (156) and (157) should actually have been written

$$\gamma_{l\to C} \;=\; \frac{(1-\pi_{\bar l})(1-\eta_l)\prod_{l\in C'\ne C}(1-\eta_{C\to l})}{\pi_{\bar l}+(1-\pi_{\bar l})(1-\eta_l)\prod_{l\in C'\ne C}(1-\eta_{C\to l})} \quad\text{and}\quad \eta'_{C\to l} \;=\; \prod_{C\ni l'\ne l}\gamma_{l'\to C},$$

to avoid division by zero. Suggest an efficient way to implement these calculations.

360. [*M23*] Find all fixed points of the seven-clause system illustrated in (159), given that $\pi_1=\pi_{\bar 2}=\pi_{\bar 4}=1$. Assume also that $\eta_l\eta_{\bar l}=0$ for all l.

▶ **361.** [*M22*] Describe all fixed points $\eta_{C\to l}=\eta'_{C\to l}$ of the equations (154), (156), (157), for which each $\eta_{C\to l}$ and each η_l is either 0 or 1.

362. [*20*] Spell out the computations needed to finish Algorithm S in step S8.

▶ **363.** [*M30*] (*Lattices of partial assignments.*) A partial assignment to the variables of a satisfiability problem is called *stable* (or "valid") if it is consistent and cannot be extended by unit propagation. In other words, it's stable if and only if no clause is entirely false, or entirely false except for at most one unassigned literal. Variable x_k of a partial assignment is called *constrained* if it appears in a clause where $\pm x_k$ is true but all the other literals are false (thus its value has a "reason").

The 3^n partial assignments of an n-variable problem can be represented either as strings $x=x_1\ldots x_n$ on the alphabet $\{0,1,*\}$ or as sets L of strictly distinct literals. For example, the string $x={*}1{*}01{*}$ corresponds to the set $L=\{2,\bar4,5\}$. We write $x\prec x'$ if x' is equal to x except that $x_k=*$ and $x'_k\in\{0,1\}$; equivalently $L\prec L'$ if $L'=L\cup k$ or $L'=L\cup\bar k$. Also $x\sqsubseteq x'$ if there are $t\ge0$ stable partial assignments $x^{(j)}$ with

$$x \;=\; x^{(0)} \prec x^{(1)} \prec \cdots \prec x^{(t)} \;=\; x'.$$

lopsidependency graph
dependency graph
Clique Local Lemma
clique cover
field
Lattices of partial assignments
partial assignment
stable
valid partial assignment
consistent
unit propagation
constrained
reason
strictly distinct literals

Let $p_1, \ldots, p_n, q_1, \ldots, q_n$ be probabilities, with $p_k + q_k = 1$ for $1 \le k \le n$. Define the weight $W(x)$ of a partial assignment to be 0 if x is unstable, otherwise

$$W(x) = \prod\{p_k \mid x_k = *\} \cdot \prod\{q_k \mid x_k \ne * \text{ and } x_k \text{ is unconstrained}\}.$$

[E. Maneva, E. Mossel, and M. J. Wainwright, in *JACM* **54** (2007), 17:1–17:41, studied general message-passing algorithms on partial assignments that are distributed with probability proportional to their weights, in the case $p_1 = \cdots = p_n = p$, showing that survey propagation (Algorithm S) corresponds to the limit as $p \to 1$.]

a) True or false: The partial assignment specified by the literals currently on the trail in step C5 of Algorithm C is stable.
b) What weights $W(x)$ correspond to the clauses F in (1)?
c) Let x be a stable partial assignment with $x_k = 1$, and let x' and x'' be obtained from x by setting $x'_k \leftarrow 0$, $x''_k \leftarrow *$. True or false: x_k is unconstrained in x if and only if (i) x' is consistent; (ii) x' is stable; (iii) x'' is stable.
d) If the only clause is $123 = (x_1 \vee x_2 \vee x_3)$, find all sets L such that $L \sqsubseteq \{1, \bar{2}, \bar{3}\}$.
e) What are the weights when there's only a single clause $123 = (x_1 \vee x_2 \vee x_3)$?
f) Find clauses such that the sets L with $L \sqsubseteq \{1,2,3,4,5\}$ are $\emptyset$, $\{4\}$, $\{5\}$, $\{1,4\}$, $\{2,5\}$, $\{4,5\}$, $\{1,4,5\}$, $\{2,4,5\}$, $\{3,4,5\}$, $\{1,3,4,5\}$, $\{2,3,4,5\}$, $\{1,2,3,4,5\}$.
g) Let $\mathcal{L}$ be a family of sets $\subseteq \{1,\ldots,n\}$, closed under intersection, with the property that $L \in \mathcal{L}$ implies $L = L^{(0)} \prec L^{(1)} \prec \cdots \prec L^{(t)} = \{1,\ldots,n\}$ for some $L^{(j)} \in \mathcal{L}$. (The sets in (d) form one such family, with $n = 5$.) Construct strict Horn clauses with the property that $L \in \mathcal{L}$ if and only if $L \sqsubseteq \{1,\ldots,n\}$.
h) True or false: If L, L', L'' are stable and $L' \prec L$, $L'' \prec L$, then $L' \cap L''$ is stable.
i) If $L' \sqsubseteq L$ and $L'' \sqsubseteq L$, prove that $L' \cap L'' \sqsubseteq L$.
j) Prove that $\sum_{x' \sqsubseteq x} W(x') = \prod\{p_k \mid x_k = *\}$ whenever x is stable.

▸ **364.** [*M21*] A *covering assignment* is a stable partial assignment in which every assigned variable is constrained. A *core assignment* is a covering assignment L that satisfies $L \sqsubseteq L'$ for some total assignment L'.

a) True or false: The empty partial assignment $L = \emptyset$ is always covering.
b) Find all the covering and core assignments of the clauses F in (1).
c) Find all the covering and core assignments of the clauses R' in (7).
d) Show that every satisfying assignment L' has a unique core.
e) The satisfying assignments form a graph, if two of them are adjacent when they differ by complementing just one literal. The connected components of this graph are called *clusters*. Prove that the elements of each cluster have the same core.
f) If L' and L'' have the same core, do they belong to the same cluster?

365. [*M27*] Prove that the clauses *waerden*$(3,3;n)$ have a nontrivial (i.e., nonempty) covering assignment for all sufficiently large n (although they're unsatisfiable).

▸ **366.** [*18*] Preprocess the clauses R' of (7). What erp rules are generated?

▸ **367.** [*20*] Justify the erp rule (161) for elimination by resolution.

368. [*16*] Show that subsumption and downhill resolution imply unit conditioning: Any preprocessor that does transformations 2 and 4 will also do transformation 1.

▸ **369.** [*21*] (N. Eén and A. Biere.) Suppose l appears only in clauses $C_1, \ldots, C_p$ and $\bar{l}$ appears only in clauses $C'_1, \ldots, C'_q$, where we have $C_1 = (l \vee l_1 \vee \cdots \vee l_r)$ and $C'_j = (\bar{l} \vee \bar{l}_j)$ for $1 \le j \le r$. Prove that we can eliminate $|l|$ by using the erp rule $\bar{l} \leftarrow (l_1 \vee \cdots \vee l_r)$ and replacing those $p+q$ clauses by only $(p-2)r + q$ others, namely

$$\{C_1 \diamond C'_j \mid r < j \le q\} \;\cup\; \{C_i \diamond C'_j \mid 1 < i \le p, 1 \le j \le r\}.$$

Maneva
Mossel
Wainwright
message-passing algorithms
survey propagation
trail
Horn clauses
covering assignment
core assignment
empty partial assignment
satisfying assignment
clusters
waerden
Preprocess
erp rules
elimination by resolution
downhill resolution
unit conditioning
Eén
Biere
erp rule

(The case $r = 1$ is especially important. In many applications — for example in the examples of fault testing, tomography, and the "Life in 4" problem about extending Fig. 35 — more than half of all variable eliminations admit this simplification.)

370. [*20*] The clauses obtained by resolution might be needlessly complex even when exercise 369 doesn't apply. For example, suppose that variable x appears only in the clauses $(x \vee a) \wedge (x \vee \bar{a} \vee c) \wedge (\bar{x} \vee b) \wedge (\bar{x} \vee \bar{b} \vee \bar{c})$. Resolution replaces those four clauses by three others: $(a \vee b) \wedge (a \vee \bar{b} \vee \bar{c}) \wedge (\bar{a} \vee b \vee c)$. Show, however, that only *two* clauses, both *binary*, would actually suffice in this particular case.

371. [*24*] By preprocessing repeatedly with transformations 1–4, and using exercise 369, prove that the 32 clauses (9) of $waerden(3,3;9)$ are unsatisfiable.

372. [*25*] Find a "small" set of clauses that *cannot* by solved entirely via transformations 1–4 and the use of exercise 369.

373. [*25*] The answer to exercise 228 defines $2m + \sum_{j=1}^{m}(j-1)^2 \approx m^3/3$ clauses in m^2 variables that suffice to refute the anti-maximal-element axioms of (99)–(101). Algorithm L needs exponential time to handle these clauses, according to Theorem R; and experiments show that they are bad news for Algorithm C too. Show, however, that preprocessing with transformations 1–4 will rapidly prove them unsatisfiable.

▸ **374.** [*32*] Design data structures for the efficient representation of clauses within a SAT preprocessor. Also design algorithms that (a) resolve clauses C and C' with respect to a variable x; (b) find all clauses C' that are subsumed by a given clause C; (c) find all clauses C' that are self-subsumed by a given clause C and a literal $l \in C$.

375. [*21*] Given $|l|$, how can one test efficiently whether or not the special situation in exercise 369 applies, using (and slightly extending) the data structures of exercise 374?

▸ **376.** [*32*] After a preprocessor has found a transformation that reduces the current set of clauses, it is supposed to try again and look for further simplifications. (See (160).) Suggest methods that will avoid unnecessary repetition of previous work, by using (and slightly extending) the data structures of exercise 374.

377. [*22*] (V. Vassilevska Williams.) If G is a graph with n vertices and m edges, construct a 2SAT problem F with $3n$ variables and $6m$ clauses, such that G contains a triangle (a 3-clique) if and only if F has a failed literal.

378. [*20*] (*Blocked clause elimination.*) Clause $C = (l \vee l_1 \vee \cdots \vee l_q)$ is said to be blocked by the literal l if every clause that contains $\bar{l}$ also contains either $\bar{l}_1$ or $\cdots$ or $\bar{l}_q$. Exercise 161(b) proves that clause C can be removed without making an unsatisfiable problem satisfiable. Show that this transformation requires an erp rule, even though it doesn't eliminate any of the variables. What erp rule works?

▸ **379.** [*20*] (*Blocked self-subsumption.*) Consider the clause $(a \vee b \vee c \vee d)$, and suppose that every clause containing $\bar{a}$ but not $\bar{b}$ nor $\bar{c}$ also contains d. Show that we can then shorten the clause to $(b \vee c \vee d)$ without affecting satisfiability. Is an erp rule needed?

380. [*21*] Sometimes we can use self-subsumption backwards, for example by weakening the clause $(l_1 \vee l_2 \vee l_3)$ to $(l_1 \vee \cdots \vee l_k)$ if each intermediate replacement of $(l_1 \vee \cdots \vee l_j)$ by $(l_1 \vee \cdots \vee l_{j-1})$ is justifiable for $3 < j \le k$. Then, if we're lucky, the clause $(l_1 \vee \cdots \vee l_k)$ is weak enough to be eliminated; in such cases we are allowed to eliminate $(l_1 \vee l_2 \vee l_3)$.

a) Show that $(a \vee b \vee c)$ can be eliminated if it is accompanied by the additional clauses $(a \vee b \vee \bar{d})$, $(a \vee d \vee e)$, $(b \vee d \vee \bar{e})$.

b) Show that $(a \vee b \vee c)$ can also be eliminated when accompanied by $(a \vee b \vee \bar{d})$, $(a \vee \bar{c} \vee \bar{d})$, $(b \vee d \vee \bar{e})$, $(b \vee \bar{c} \vee \bar{e})$, provided that no other clauses contain $\bar{c}$.

fault testing
tomography
Life in 4
resolution
waerden
anti-maximal-element
data structures
preprocessor
resolution, implementation of
subsumption, implementation of
self-subsumed
Vassilevska Williams
clique
failed literal
triangle-free graph
Blocked clause elimination
erp rule
Blocked self-subsumption
self-subsumption

c) What erp rules, if any, are needed for those eliminations?

381. [*22*] Combining exercises 379 and 380, show that any one of the clauses in

$$(\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3) \wedge \cdots \wedge (\bar{x}_{n-1} \vee x_n) \wedge (\bar{x}_n \vee x_1)$$

can be removed if there are no other clauses with negative literals. State the erp rules.

382. [*30*] Although the techniques in the preceding exercises are computationally difficult to apply, show that a lookahead forest based on the dependency digraph can be used to discover some of those simplifications efficiently.

▸ **383.** [*23*] (*Inprocessing.*) A SAT solver can partition its database of current clauses into two parts, the "hard" clauses Φ and the "soft" clauses Ψ. Initially Ψ is empty, while Φ is F, the set of all input clauses. Four kinds of changes are subsequently allowed:

- **Learning.** We can append a new soft clause C, provided that $\Phi \cup \Psi \cup C$ is satisfiable whenever $\Phi \cup \Psi$ is satisfiable.
- **Forgetting.** We can discard (purge) any soft clause.
- **Hardening.** We can reclassify any soft clause and call it hard.
- **Softening.** We can reclassify any hard clause C and call it soft, provided that Φ is satisfiable whenever $\Phi \setminus C$ is satisfiable. In this case we also should output any necessary erp rules, which change the settings of variables in such a way that any solution to $\Phi \setminus C$ becomes a solution to Φ.

a) Prove that, throughout any such procedure, F is satisfiable $\iff$ Φ is satisfiable $\iff$ $\Phi \cup \Psi$ is satisfiable.

b) Furthermore, given any solution to Φ, we obtain a solution to F by applying the erp rules in reverse order.

c) What is wrong with the following scenario? Start with one hard clause, (x), and no soft clauses. Reclassify (x) as soft, using the erp rule $x \leftarrow 1$. Then append a new soft clause $(\bar{x})$.

d) If C is certifiable for Φ (see exercise 385), can we safely learn C?

e) If C is certifiable for $\Phi \setminus C$, can we safely forget C?

f) In what cases is it legitimate to discard a clause, hard or soft, that is subsumed by another clause, hard or soft?

g) In what cases is self-subsumption permissible?

h) Explain how to eliminate all clauses that involve a particular variable x.

i) Show that, if z is a new variable, we can safely learn the three new soft clauses $(x \vee z)$, $(y \vee z)$, $(\bar{x} \vee \bar{y} \vee \bar{z})$ in Tseytin's concept of extended resolution.

384. [*25*] Continuing the previous exercise, show that we can always safely forget any clause C that contains a literal l for which $C \diamond C'$ is certifiable for $\Phi \setminus C$ whenever $C' \in \Phi$ contains $\bar{l}$. What erp rule is appropriate?

385. [*22*] Clause C is called *certifiable* for a set of clauses F if $F \wedge \overline{C} \vdash_1 \epsilon$, as in (119). It is said to be *absorbed* by F if it is nonempty and $F \wedge \overline{C \setminus l} \vdash_1 l$ for every $l \in C$, or if it is empty and $F \vdash_1 \epsilon$. (Every clause of F is obviously absorbed by F.)

a) True or false: If C is absorbed by F, it is certifiable for F.

b) Which of $\{\bar{1}, \bar{1}2, \bar{1}23\}$ are implied by, certifiable for, or absorbed by R' in (7)?

c) If C is certifiable for F and if all clauses of F are absorbed by F', prove that C is certifiable for F'.

d) If C is absorbed by F and if all clauses of F are absorbed by F', prove that C is absorbed by F'.

lookahead forest
dependency digraph
Inprocessing
hard
soft
Learning
Forgetting
purge
erp rules
certifiable
subsumed
self-subsumption
variable elimination
Tseytin
extended resolution
erp rule
certifiable
absorbed
asymmetric tautology, see certifiable clause

▸ **386.** [*M25*] Let Algorithm C_0 be a variant of Algorithm C that (i) makes all decisions at random; (ii) never forgets a learned clause; and (iii) restarts whenever a new clause has been learned. (Thus, step C5 ignores M_p and M_f; step C6 chooses l uniformly at random from among the $2(n-F)$ currently unassigned literals; step C8 backjumps while $F > i_1$, instead of while $F > i_{d'+1}$; and after step C9 has stored a new clause, with $d > 0$, it simply sets $d \leftarrow 0$ and returns to C2. The data structures `HEAP`, `OVAL`, and `ACT` are no longer used.) We will prove that Algorithm C_0 is, nevertheless, quite powerful.

In the remainder of this exercise, F denotes the set of clauses known by Algorithm C_0, both original and learned; in particular, the unit clauses of F will be the first literals $L_0, L_1, \ldots, L_{i_1-1}$ on the trail. If C is any clause and if $l \in C$, we define

$$\operatorname{score}(F,C,l) = \begin{cases} \infty, & \text{if } F \wedge \overline{C \setminus l} \vdash_1 l; \\ \bigl|\{l' \mid F \wedge \overline{C \setminus l} \vdash_1 l'\}\bigr|, & \text{otherwise.} \end{cases}$$

Thus score(F,C,l) represents the total number of literals on the trail after making all the unforced decisions of $\overline{C} \setminus \bar{l}$, if no conflict arises. We say that Algorithm C_0 performs a "helpful round" for C and l if (i) every decision literal belongs to $\overline{C}$; and (ii) $\bar{l}$ is chosen as a decision literal only if the other elements of $\overline{C}$ are already in the trail.

a) Let C be certifiable for F, and suppose that score$(F,C,l) < \infty$ for some $l \in C$. Prove that if F' denotes F together with a clause learned on a helpful round, then score$(F',C,l) >$ score(F,C,l).
b) Furthermore score$(F',C,l) \geq$ score(F,C,l) after an unhelpful round.
c) Therefore C will be absorbed by the set F' of known clauses after at most $|C|n$ helpful rounds have occurred.
d) If $|C| = k$, show that Pr(helpful round) $\geq (k-1)!/(2n)^k \geq 1/(4n^k)$.
e) Consequently, by exercise 385(c), if there exists a certificate of unsatisfiability $(C_1, \ldots, C_t)$ for a family of clauses F with n variables, Algorithm C_0 will prove F unsatisfiable after learning an average of $\mu \leq 4\sum_{i=1}^{t} |C_i| n^{1+|C_i|}$ clauses. (And it will q.s. need to learn at most $\mu \ln n \ln\ln n$ clauses, by exercise MPR–102.)

▸ **387.** [*21*] Graph G is said to be *embedded* in graph G' if every vertex v of G corresponds to a distinct vertex v' of G', where $u' — v'$ in G' whenever $u — v$ in G. Explain how to construct clauses that are satisfiable if and only if G can be embedded in G'.

388. [*20*] Show that the problems of deciding whether or not a given graph G (a) contains a k-clique, (b) can be k-colored, or (c) has a Hamiltonian cycle can all be regarded as graph embedding problems.

▸ **389.** [*22*] In this 4×4 diagram, it's possible to trace out the phrase '`THE␣ART␣OF␣COMPUTER␣PROGRAMMING`' by making only king moves and knight moves, *except* for the final step from `N` to `G`.

N	T	E	F
H	I	R	␣
U	P	O	A
M	M	C	G

Rearrange the letters so that the entire phrase can be traced.

▸ **390.** [*23*] Let G be a graph with vertices V, edges E, $|E| = m$, $|V| = n$, and $s, t \in V$.
a) Construct $O(kn)$ clauses that are satisfiable if and only if there's a path of length k or less from s to t, given k.
b) Construct $O(m)$ clauses that are satisfiable if and only if there's at least one path from s to t.
c) Construct $O(n^2)$ clauses that are satisfiable if and only if G is connected.
d) Construct $O(km)$ clauses that are *un*satisfiable if and only if there's a path of length k or less from s to t, given k.

helpful round
certificate of unsatisfiability
embedded
homomorphic embedding
clique
Hamiltonian cycle
king moves
knight moves
path
reachability in a graph
connected
path

e) Construct $O(m)$ clauses that are *un*satisfiable if and only if there's at least one path from s to t.

f) Construct $O(m)$ clauses that are *un*satisfiable if and only if G is connected. (This construction is much better than (c), in a sparse graph.)

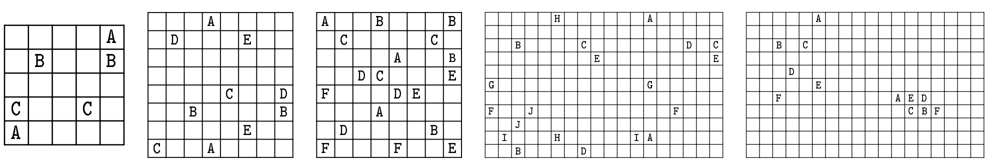
391. [*M25*] The values of two integer variables satisfy $0 \le x, y < d$, and they are to be represented as l-bit quantities $x_{l-1} \ldots x_0$, $y_{l-1} \ldots y_0$, where $l = \lceil \lg d \rceil$. Specify three different ways to encode the relation $x \ne y$:

a) Let $x = (x_{l-1} \ldots x_0)_2$ and $y = (y_{l-1} \ldots y_0)_2$; and let the encoding enforce the conditions $(x_{l-1} \ldots x_0)_2 < d$, and $(y_{l-1} \ldots y_0)_2 < d$, as well as ensuring that $x \ne y$ by introducing $2l + 1$ additional clauses in l auxiliary variables.

b) Like (a), but there are d additional clauses (not $2l + 1$), and no auxiliaries.

c) All bit patterns $x_{l-1} \ldots x_0$ and $y_{l-1} \ldots y_0$ are valid, but some values might have two different patterns. The encoding has d clauses and no auxiliary variables.

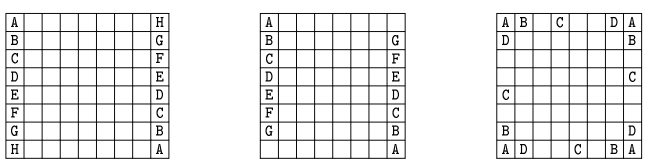
392. [*22*] The blank spaces in the following diagrams can be filled with letters in such a way that all occurrences of the same letter are rookwise connected:



(*i*) (*ii*) (*iii*) (*iv*) (*v*)

a) Demonstrate how to do it. (Puzzle (*i*) is easy; the others less so.)

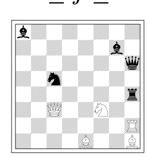
b) Similarly, solve the following puzzles — but use *kingwise* connectedness instead.



(*vi*) (*vii*) (*viii*)

c) Construct clauses with which a SAT solver can solve general puzzles of this kind: Given a graph G and disjoint sets of vertices $T_1, T_2, \ldots, T_t$, a solution should exhibit disjoint *connected* sets of vertices $S_1, S_2, \ldots, S_t$, with $T_j \subseteq S_j$ for $1 \le j \le t$.

393. [*25*] (T. R. Dawson, 1911.) Show that it's possible for each white piece in the accompanying chess diagram to capture the corresponding black piece, via a path that doesn't intersect any of the other paths. How can SAT help to solve this problem?



394. [*25*] One way to encode the at-most-one constraint $S_{\le 1}(y_1, \ldots, y_p)$ is to introduce $l = \lceil \lg p \rceil$ auxiliary variables together with the following $nl + n - 2^l$ clauses, which essentially "broadcast" the value of j when y_j becomes true:

$$(\bar{y}_j \vee (-1)^{b_t} a_t) \quad \text{for } 1 \le j \le p,\ 1 \le t \le q = \lfloor \lg(2p - j) \rfloor, \text{ where } 2p - j = (1b_1 \ldots b_q)_2.$$

For example, the clauses when $p = 3$ are $(\bar{y}_1 \vee a_1) \wedge (\bar{y}_1 \vee \bar{a}_2) \wedge (\bar{y}_2 \vee a_1) \wedge (\bar{y}_2 \vee a_2) \wedge (\bar{y}_3 \vee \bar{a}_1)$.

Experiment with this encoding by applying it to Langford's problem, using it in place of (13) whenever $p \ge 7$.

395. [*20*] What clauses should replace (15), (16), and (17) if we want to use the order encoding for a graph coloring problem?

connected
encode
rookwise connected
multicommodity flow
routing, disjoint
connection puzzles
kingwise connectedness
Dawson
chess diagram
nonintersecting paths
at-most-one
auxiliary variables
broadcast
Langford's problem
order encoding

▸ **396.** [*23*] (*Double clique hints.*) If x has one of the d values $\{0,1,\ldots,d-1\}$, we can represent it binarywise with respect to *two* different orderings by letting $x^j=[x\geq j]$ and $\hat{x}^j=[x\pi\geq j]$ for $1\leq j<d$, where π is any given permutation. For example, if $d=4$ and $(0\pi,1\pi,2\pi,3\pi)=(2,3,0,1)$, the representations $x^1x^2x^3{:}\hat{x}^1\hat{x}^2\hat{x}^3$ of 0, 1, 2, and 3 are respectively 000:110, 100:111, 110:000, and 111:100. This double ordering allows us to encode graph coloring problems by including not only the hints (162) but also

$$(\overline{\hat{v}_1^{d-k+1}}\vee\cdots\vee\overline{\hat{v}_k^{d-k+1}})\wedge(\hat{v}_1^{k-1}\vee\cdots\vee\hat{v}_k^{k-1}),$$

whenever the vertices $\{v_1,\ldots,v_k\}$ form a k-clique.

Explain how to construct clauses for this encoding, and experiment with coloring the $n\times n$ queens graph when $(0\pi,1\pi,2\pi,3\pi,4\pi,\ldots)=(0,d{-}1,1,d{-}2,2,\ldots)$ is the inverse of the organ-pipe permutation.

▸ **397.** [*22*] (N. Tamura, 2014.) Suppose $x_0, x_1, \ldots, x_{p-1}$ are integer variables with the range $0\leq x_i<d$, represented in order encoding by Boolean variables $x_i^j=[x_i\geq j]$ for $0\leq i<p$ and $1\leq j<d$. Show that the *all-different* constraint, "$x_i\neq x_j$ for $0\leq i<j<p$," can be nicely encoded by introducing auxiliary integer variables $y_0, y_1, \ldots, y_{d-1}$ with the range $0\leq y_j<p$, represented in order encoding by Boolean variables $y_j^i=[y_j\geq i]$ for $1\leq i<p$ and $0\leq j<d$, and by devising clauses to enforce the condition $x_i=j \implies y_j=i$. Furthermore, hints analogous to (162) can be given.

398. [*18*] Continuing exercise 397, what's an appropriate way to enforce the all-different constraint when $x_0, \ldots, x_{p-1}$ are represented in the *direct* encoding?

▸ **399.** [*23*] If the variables u and v range over d values $\{1,\ldots,d\}$, it's natural to encode them directly as sequences $u_1\ldots u_d$ and $v_1\ldots v_d$, where $u_i=[u=i]$ and $v_j=[v=j]$, using the at-least-one clauses (15) and the at-most-one clauses (17). A *binary constraint* tells us which pairs (i,j) are legal; for example, the graph-coloring constraint says that $i\neq j$ when i and j are the colors of adjacent vertices in some graph.

One way to specify such a constraint is to assert the *preclusion clauses* $(\bar{u}_i\vee\bar{v}_j)$ for all *illegal* pairs (i,j), as we did for graph coloring in (16). But there's also another general way: We can assert the *support clauses*

$$\bigwedge_{i=1}^{d}\Bigl(\bar{u}_i\vee\bigvee\{v_j\mid(i,j)\text{ is legal}\}\Bigr)\;\wedge\;\bigwedge_{j=1}^{d}\Bigl(\bar{v}_j\vee\bigvee\{u_i\mid(i,j)\text{ is legal}\}\Bigr)$$

instead. Graph coloring with d colors would then be represented by clauses such as $(\bar{u}_3\vee v_1\vee v_2\vee v_4\vee\cdots\vee v_d)$, when u and v are adjacent.

a) Suppose t of the d^2 pairs (i,j) are legal. How many preclusion clauses are needed? How many support clauses?

b) Prove that the support clauses are always at least as strong as the preclusion clauses, in the sense that all consequences of the preclusion clauses under unit propagation are also consequences of the support clauses under unit propagation, given any partial assignment to the binary variables $\{u_1,\ldots,u_d,v_1,\ldots,v_d\}$.

c) Conversely, in the case of the graph-coloring constraint, the preclusion clauses are also at least as strong as the support clauses (hence equally strong).

d) However, exhibit a binary constraint for which the support clauses are strictly stronger than the preclusion clauses.

400. [*25*] Experiment with preclusion clauses versus support clauses by applying them to the n queens problem. Use Algorithms L, C, and W for comparison.

401. [*16*] If x has the unary representation $x^1x^2\ldots x^{d-1}$, what is the unary representation of (a) $y=\lceil x/2\rceil$? (b) $z=\lfloor(x+1)/3\rfloor$?

Double clique hints
order encoding
graph coloring
hints
redundant representation
clique
queens graph
organ-pipe permutation
Tamura
order encoding
all-different
direct encoding
direct encoding
at-least-one
at-most-one
binary constraint
graph-coloring
preclusion clauses
conflict clauses, see also preclusion clauses
support clauses
unit propagation
n queens problem
unary representation

402. [*18*] If x has the unary representation $x^1x^2\ldots x^{d-1}$, encode the further condition that x is (a) even; (b) odd.

403. [*20*] Suppose x, y, z have the order encoding, with $0 \le x, y, z < d$. What clauses enforce (a) $\min(x,y) \le z$? (b) $\max(x,y) \le z$? (c) $\min(x,y) \ge z$? (d) $\max(x,y) \ge z$?

▸ **404.** [*21*] Continuing exercise 403, encode the condition $|x-y| \ge a$, for a given constant $a \ge 1$, using either (a) d clauses of length ≤ 4 and no auxiliary variables; or (b) $2d - O(a)$ clauses of length ≤ 3, and one auxiliary variable.

▸ **405.** [*M23*] The purpose of this exercise is to encode the constraint $ax + by \le c$, when a, b, c are integer constants, assuming that x, y are order-encoded with range $[0\,.\,.\,d)$.

a) Prove that it suffices to consider cases where $a, b, c > 0$.
b) Exhibit a suitable encoding for the special case $13x - 8y \le 7$, $d = 8$.
c) Exhibit a suitable encoding for the special case $13x - 8y \ge 1$, $d = 8$.
d) Specify an encoding that works for general a, b, c, d.

406. [*M24*] Order-encode (a) $xy \le a$ and (b) $xy \ge a$, when a is an integer constant.

▸ **407.** [*M22*] If x, y, z are order-encoded, with $0 \le x, y < d$ and $0 \le z < 2d-1$, the clauses

$$\bigwedge_{k=1}^{2d-2} \; \bigwedge_{j=\max(0,k+1-d)}^{k} (\bar{x}^j \vee \bar{y}^{k-j} \vee z^k)$$

are satisfiable if and only if $x+y \le z$; this is the basic idea underlying (20). Another way to encode the same relation is to introduce new order-encoded variables u and v, and to construct clauses for the relations $\lfloor x/2\rfloor + \lfloor y/2\rfloor \le u$ and $\lceil x/2\rceil + \lceil y/2\rceil \le v$, recursively using methods for numbers less than $\lceil d/2\rceil$ and $\lfloor d/2\rfloor + 1$. Then we can finish the job by letting $z^1 = v^1$, $z^{2d-2} = v^d$ (d even) or u^{d-1} (d odd), and appending the clauses

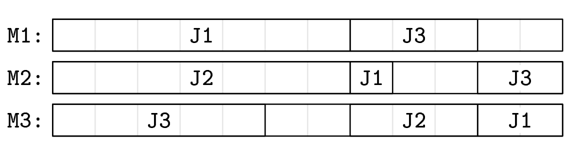
$$(\bar{u}^j \vee z^{2j}) \wedge (\bar{v}^{j+1} \vee z^{2j}) \wedge (\bar{u}^j \vee \bar{v}^{j+1} \vee z^{2j+1}), \quad \text{for } 1 \le j \le d-2.$$

a) Explain why the alternative method is valid.
b) For what values of d does that method produce fewer clauses?
c) Consider analogous methods for the relation $x + y \ge z$.

▸ **408.** [*25*] (*Open shop scheduling.*) Consider a system of m machines and n jobs, together with an $m \times n$ matrix of nonnegative integer weights $W = (w_{ij})$ that represent the amount of uninterrupted time on machine i that is needed by job j.

The open shop scheduling problem seeks a way to get all the work done in t units of time, without assigning two jobs simultaneously to the same machine and without having two machines simultaneously assigned to the same job. We want to minimize t, which is called the "makespan" of the schedule.

For example, suppose $m = n = 3$ and $W = \begin{pmatrix} 703 \\ 172 \\ 235 \end{pmatrix}$. A "greedy" algorithm, which repeatedly fills the lexicographically smallest time slot (t, i, j) such that $w_{ij} > 0$ but neither machine i nor job j have yet been scheduled at time t, achieves a makespan of 12 with the following schedule:



a) Is 12 the optimum makespan for this W?

auxiliary variables
linear inequalities
recursively
Open shop scheduling
job shop problems
makespan
"greedy" algorithm

b) Prove that the greedy algorithm always produces a schedule whose makespan is less than $(\max_{i=1}^{m}\sum_{j=1}^{n} w_{ij}) + (\max_{j=1}^{n}\sum_{i=1}^{m} w_{ij})$, unless W is entirely zero.

c) Suppose machine i begins to work on job j at time s_{ij}, when $w_{ij} > 0$. What conditions should these starting times satisfy, in order to achieve the makespan t?

d) Show that the *order encoding* of these variables s_{ij} yields SAT clauses that nicely represent any open shop scheduling problem.

e) Let $\lfloor W/k \rfloor$ be the matrix obtained by replacing each element w_{ij} of W by $\lfloor w_{ij}/k \rfloor$. Prove that if the open shop scheduling problem for $\lfloor W/k \rfloor$ and t is unsatisfiable, so is the problem for W and kt.

▸ **409.** [*M26*] Continuing exercise 408, find the best makespans in the following cases:

a) $m = 3$, $n = 3r + 1$; $w_{1j} = w_{2(r+j)} = w_{3(2r+j)} = a_j$ for $1 \le j \le r$; $w_{1n} = w_{2n} = w_{3n} = \lfloor (a_1+\cdots+a_r)/2 \rfloor$; otherwise $w_{ij} = 0$. (The positive integers a_j are given.)

b) $m = 4$, $n = r + 2$; $w_{1j} = (r + 1)a_j$ and $w_{2j} = 1$ for $1 \le j \le r$; $w_{2(n-1)} = w_{2n} = (r + 1)\lfloor (a_1 + \cdots + a_r)/2 \rfloor$; $w_{3(n-1)} = w_{4n} = w_{2n} + r$; otherwise $w_{ij} = 0$.

c) $m = n$; $w_{jj} = n - 2$, $w_{jn} = w_{nj} = 1$ for $1 \le j < n$; otherwise $w_{ij} = 0$.

d) $m = 2$; $w_{1j} = a_j$ and $w_{2j} = b_j$ for $1 \le j \le n$, where $a_1+\cdots+a_n = b_1+\cdots+b_n = s$ and $a_j + b_j \le s$ for $1 \le j \le n$.

410. [*24*] Exhibit clauses for the constraint $13x - 8y \le 7$ when x and y are *log-encoded* as 3-bit integers $x = (x_2x_1x_0)_2$ and $y = (y_2y_1y_0)_2$. (Compare with exercise 405(b).)

▸ **411.** [*25*] If $x = (x_m \ldots x_1)_2$, $y = (y_n \ldots y_1)_2$, and $z = (z_{m+n} \ldots z_1)_2$ stand for binary numbers, the text explains how to encode the relation $xy = z$ with fewer than $20mn$ clauses, using Napier–Dadda multiplication. Explain how to encode the relations $xy \le z$ and $xy \ge z$ with fewer than $9mn$ and $11mn$ clauses, respectively.

412. [*40*] Experiment with the encoding of somewhat large numbers by using a radix-d representation in which each digit has the order encoding.

413. [*M20*] How many clauses will remain after the auxiliary variables $a_1, \ldots, a_{n-1}$ of (169) have been eliminated by resolution?

▸ **414.** [*M22*] Generalize (169) to an encoding of lexicographic order on d-ary vectors, $(x_1 \ldots x_n)_d \le (y_1 \ldots y_n)_d$, where each $x_k = x_k^1 + \cdots + x_k^{d-1}$ and $y_k = y_k^1 + \cdots + y_k^{d-1}$ has the order encoding. What modifications to your construction will encode the *strict* relation $x_1 \ldots x_n < y_1 \ldots y_n$?

415. [*M22*] Find all CNF formulas for the function $(x_1 \oplus y_1) \vee \cdots \vee (x_n \oplus y_n)$.

416. [*20*] Encode the condition 'if $x_1 \ldots x_n = y_1 \ldots y_n$ then $u_1 \ldots u_m = v_1 \ldots v_m$', using $2m+2n+1$ clauses and $n+1$ auxiliary variables. *Hint:* $2n$ of the clauses are in (172).

417. [*21*] Continuing exercise 42, what is the Tseytin encoding of the ternary mux operation '$s \leftarrow t?\, u{:}\, v$'? Use it to justify the translation of branching programs via (174).

418. [*23*] Use a branching program to construct clauses that are satisfiable if and only if (x_{ij}) is an $m \times n$ Boolean matrix whose rows satisfy the hidden weighted bit function h_n and whose columns satisfy the complementary function $\bar{h}_m$. In other words,

$$r_i = \sum_{j=1}^{n} x_{ij}, \;\; c_j = \sum_{i=1}^{m} x_{ij}, \;\text{ and }\; x_{ir_i} = 1, \; x_{c_jj} = 0, \;\text{ assuming that } x_{i0} = x_{0j} = 0.$$

419. [*M21*] If $m, n \ge 3$, find (by hand) all solutions to the problem of exercise 418 such that (a) $\sum x_{ij} = m + 1$ (the minimum); (b) $\sum x_{ij} = mn - n - 1$ (the maximum).

420. [*18*] Derive (175) mechanically (that is, "without thinking") from the Boolean chain $s \leftarrow x_1 \oplus x_2$, $c \leftarrow x_1 \wedge x_2$, $t \leftarrow s \oplus x_3$, $c' \leftarrow s \wedge x_3$, requiring $c = c' = 0$.

order encoding
linear inequalities
log-encoded
Napier
Dadda
multiplication
radix-d representation
order encoding
auxiliary variables
eliminated
resolution
lexicographic order
order encoding
CNF
Tseytin encoding
mux operation
branching programs
hidden weighted bit function
Boolean chain

421. [*18*] Derive (176) mechanically from the branching program $I_5 = (\bar{1}?\,4{:}\,3)$, $I_4 = (\bar{2}?\,1{:}\,2)$, $I_3 = (\bar{2}?\,2{:}\,0)$, $I_2 = (\bar{3}?\,1{:}\,0)$, beginning at I_5.

422. [*11*] What does unit propagation deduce when the additional clause (x_1) or (x_2) is appended to (a) F in (175)? (b) G in (176)?

423. [*22*] A representation F that satisfies a condition like (180) but with l replaced by ϵ can be called "weakly forcing." Exercise 422 shows that (175) and (176) are weakly forcing. Does the BDD of *every* function define a weakly forcing encoding, via (173)?

▶ **424.** [*20*] The dual of the Pi function has the prime clauses $\{\bar{1}\bar{2}\bar{3}, \bar{1}\bar{3}\bar{4}, 2\bar{3}\bar{4}, 234, 12\}$ (see 7.1.1–(30)). Can any of them be omitted from a forcing representation?

425. [*18*] A clause with exactly one positive literal is called a definite Horn clause, and Algorithm 7.1.1C computes the "core" of such clauses. If F consists of definite Horn clauses, prove that x is in the core if and only if $F \vdash_1 x$, if and only if $F \wedge (\bar{x}) \vdash_1 \epsilon$.

▶ **426.** [*M20*] Suppose F is a set of clauses that represent $f(x_1, \ldots, x_n)$ using auxiliary variables $\{a_1, \ldots, a_m\}$ as in (170), where $m > 0$. Let G be the clauses that result after variable a_m has been eliminated as in (112).

a) True or false: If F is forcing then G is forcing.

b) True or false: If F is not forcing then G is not forcing.

427. [*M30*] Exhibit a function $f(x_1, \ldots, x_n)$ for which every set of forcing clauses that uses no auxiliary variables has size $\Omega(3^n/n^2)$, although f can actually be represented by a *polynomial* number of forcing clauses when auxiliary variables are introduced. *Hint:* See exercise 7.1.1–116.

428. [*M27*] A generic graph G on vertices $\{1, \ldots, n\}$ can be characterized by $\binom{n}{2}$ Boolean variables $X = \{x_{ij} \mid 1 \le i < j \le n\}$, where $x_{ij} = [i \text{---} j \text{ in } G]$. Properties of G can therefore be regarded as Boolean functions, $f(X)$.

a) Let $f_{nd}(X) = [\chi(G) \le d]$; that is, f_{nd} is true if and only if G has a d-coloring. Construct clauses F_{nd} that represent the function $f_{nd}(X) \vee y$, using auxiliary variables $Z = \{z_{jk} \mid 1 \le j \le n,\ 1 \le k \le d\}$ that mean "vertex j has color k."

b) Let G_{nd} be a forcing representation of the Boolean function $F_{nd}(X, y, Z)$, and suppose that G_{nd} has M clauses in N variables. (These N variables should include the $\binom{n}{2} + 1 + nd$ variables of F_{nd}, along with an arbitrary number of additional auxiliaries.) Explain how to construct a monotone Boolean chain of cost $O(MN^2)$ for the function $\bar{f}_{nd}$ (see exercise 7.1.2–84), given the clauses of G_{nd}.

Note: Noga Alon and Ravi B. Boppana, *Combinatorica* **7** (1987), 1–22, proved that every monotone chain for this function has length $\exp \Omega((n/\log n)^{1/3})$ when $d + 1 = \lfloor (n/\lg n)^{2/3}/4 \rfloor$. Hence M and N cannot be of polynomial size.

429. [*22*] Prove that Bailleux and Boufkhad's clauses (20), (21) are forcing: If any r of the x's have been set to 1, then unit propagation will force all the others to 0.

430. [*25*] Similarly, Sinz's clauses (18) and (19) are forcing.

▶ **431.** [*20*] Construct efficient, forcing clauses for the relation $x_1 + \cdots + x_m \le y_1 + \cdots + y_n$.

432. [*24*] Exercise 404 gives clauses for the relation $|x - y| \ge a$. Are they forcing?

433. [*25*] Are the lexicographic-constraint clauses in (169) forcing?

434. [*21*] Let L_l be the language defined by the regular expression $0^*1^l0^*$; in other words, the binary string $x_1 \ldots x_n$ is in L_l if and only if it consists of zero or more 0s followed by exactly l 1s followed by zero or more 0s.

a) Explain why the following clauses are satisfiable if and only if $x_1 \ldots x_n \in L_l$: (i) $(\bar{p}_k \vee \bar{x}_k)$, $(\bar{p}_k \vee p_{k-1})$, and $(\bar{p}_{k-1} \vee x_k \vee p_k)$ for $1 \le k \le n$, also (p_0); (ii) $(\bar{q}_k \vee \bar{x}_k)$,

branching program
weakly forcing
BDD
dual
Pi function
prime clauses
forcing representation
definite Horn clause
Horn core
eliminated
auxiliary variables
generic graph
forcing
Alon
Boppana
Bailleux
Boufkhad
forcing
unit propagation
Sinz
equal sums
regular expression

$(\bar{q}_k \vee q_{k+1})$, and $(\bar{q}_{k+1} \vee x_k \vee q_k)$ for $1 \le k \le n$, also (q_{n+1}); (iii) $(\bar{r}_k \vee p_{k-1}) \wedge \bigwedge_{0 \le d < l} (\bar{r}_k \vee x_{k+d}) \wedge (\bar{r}_k \vee q_{k+l})$ for $1 \le k \le n+1-l$, also $(r_1 \vee \cdots \vee r_{n+1-l})$.

b) Show that those clauses are forcing when $l = 1$ but not when $l = 2$.

▶ **435.** [*28*] Given $l \ge 2$, construct a set of $O(n \log l)$ clauses that characterize the language L_l of exercise 434 and are forcing.

436. [*M32*] (*Nondeterministic finite-state automata.*) A regular language L on the alphabet A can be defined in the following well-known way: Let Q be a finite set of "states," and let $I \subseteq Q$ and $O \subseteq Q$ be designated "input states" and "output states." Also let $T \subseteq Q \times A \times Q$ be a set of "transition rules." Then the string $x_1 \ldots x_n$ is in L if and only if there's a sequence of states $q_0, q_1, \ldots, q_n$ such that $q_0 \in I$, $(q_{k-1}, x_k, q_k) \in T$ for $1 \le k \le n$, and $q_n \in O$.

Given such a definition, where $A = \{0, 1\}$, use auxiliary variables to construct clauses that are satisfiable if and only if $x_1 \ldots x_n \in L$. The clauses should be forcing, and there should be at most $O(n|T|)$ of them.

As an example, write out the clauses for the language $L_2 = 0^*1^20^*$ of exercise 434.

437. [*M21*] Extend exercise 436 to the general case where A has more than two letters.

438. [*21*] Construct a set of forcing clauses that are satisfiable if and only if a given binary string $x_1 \ldots x_n$ contains exactly t runs of 1s, having lengths $(l_1, l_2, \ldots, l_t)$ from left to right. (Equivalently, the string $x_1 \ldots x_n$ should belong to the language defined by the regular expression $0^*1^{l_1}0^+1^{l_2}0^+ \ldots 0^+1^{l_t}0^*$.)

▶ **439.** [*30*] Find efficient forcing clauses for the constraint that $x_1 + \cdots + x_n = t$ and that there are no two consecutive 1s. (This is the special case $l_1 = \cdots = l_t = 1$ of the previous exercise, but a much simpler construction is possible.)

440. [*M33*] Extend exercise 436 to *context free languages*, which can be defined by a set $S \subseteq N$ and by production rules U and W of the following well-known forms: $U \subseteq \{P \to a \mid P \in N,\ a \in A\}$ and $W \subseteq \{P \to QR \mid P, Q, R \in N\}$, where N is a set of "nonterminal symbols." A string $x_1 \ldots x_n$ with each $x_j \in A$ belongs to the language if and only if it can be produced from a nonterminal symbol $P \in S$.

441. [*M35*] Show that any threshold function $f(x_1, \ldots, x_n) = [w_1x_1 + \cdots + w_nx_n \ge t]$ has a forcing representation whose size is polynomial in $\log|w_1| + \cdots + \log|w_n|$.

▶ **442.** [*M27*] The unit propagation relation $\vdash_1$ can be generalized to kth order propagation $\vdash_k$ as follows: Let F be a family of clauses and let l be a literal. If $(l_1, l_2, \ldots, l_p)$ is a sequence of literals, we write $L_q^- = \{l_1, \ldots, l_{q-1}, \bar{l}_q\}$ for $1 \le q \le p$. Then

$$F \vdash_0 l \iff \epsilon \in F;$$

$$F \vdash_{k+1} l \iff F \,|\, L_1^- \vdash_k \epsilon,\ F \,|\, L_2^- \vdash_k \epsilon,\ \ldots,\ \text{and } F \,|\, L_p^- \vdash_k \epsilon$$
$$\text{for some strictly distinct literals } l_1,\ l_2,\ \ldots,\ l_p \text{ with } l_p = l;$$

$$F \vdash_k \epsilon \iff F \vdash_k l \text{ and } F \vdash_k \bar{l} \text{ for some literal } l.$$

a) Verify that $\vdash_1$ corresponds to unit propagation according to this definition.
b) Describe $\vdash_2$ informally, using the concept of "failed literals."
c) Prove that $F \vdash_k \epsilon$ or $F \vdash_k \bar{l}$ implies $F \,|\, l \ \vdash_k \epsilon$ for all literals l, and furthermore that $F \vdash_k \epsilon$ implies $F \vdash_{k+1} \epsilon$, for all $k \ge 0$.
d) True or false: $F \vdash_k l$ implies $F \vdash_{k+1} l$.
e) Let $L_k(F) = \{l \mid F \vdash_k l\}$. What is $L_k(R')$, where R' appears in (7) and $k \ge 0$?
f) Given $k \ge 1$, explain how to compute $L_k(F)$ and $F \,|\, L_k(F)$ in $O(n^{2k-1}m)$ steps, when F has m clauses in n variables.

forcing
Nondeterministic finite-state automata
finite-state automata
regular language
input states
output states
transition rules
forcing
runs of 1s
regular expression
consecutive 1s
context free languages
production rules
nonterminal symbols
threshold function
Pseudo-Boolean constraints, see threshold functi
unit propagation
propagation, kth order
failed literals

443. [*M24*] (*A hierarchy of hardness.*) Continuing the previous exercise, a family of clauses F is said to belong to class UC_k if it has the property that

$$F\,|\,L \vdash \epsilon \text{ implies } F\,|\,L \vdash_k \epsilon \qquad \text{for all sets of strictly distinct literals } L.$$

("Whenever a partial assignment yields unsatisfiable clauses, the inconsistency can be detected by kth order propagation.") And F is said to belong to class PC_k if

$$F\,|\,L \vdash l \text{ implies } F\,|\,L \vdash_k l \qquad \text{for all sets of strictly distinct literals } L \cup l.$$

a) Prove that $\mathrm{PC}_0 \subset \mathrm{UC}_0 \subset \mathrm{PC}_1 \subset \mathrm{UC}_1 \subset \mathrm{PC}_2 \subset \mathrm{UC}_2 \subset \cdots$, where the set inclusions are strict (each class is contained in but unequal to its successor).
b) Describe all families F that belong to the smallest class, PC_0.
c) Give interesting examples of families in the next smallest class, UC_0.
d) True or false: If F contains n variables, $F \in \mathrm{PC}_n$.
e) True or false: If F contains n variables, $F \in \mathrm{UC}_{n-1}$.
f) Where do the clauses R' of (7) fall in the hierarchy?

444. [*M26*] The following *single lookahead unit resolution* algorithm, called SLUR, returns either 'sat', 'unsat', or 'maybe', depending on whether a given set F of clauses is satisfiable, unsatisfiable, or beyond its ability to decide via easy propagations:

E1. [Propagate.] If $F \vdash_1 \epsilon$, terminate ('unsat'). Otherwise set $F \leftarrow F\,|\,\{l \mid F \vdash_1 l\}$.

E2. [Satisfied?] If $F = \emptyset$, terminate ('sat'). Otherwise set l to any literal within F.

E3. [Lookahead and propagate.] If $F\,|\,l \not\vdash_1 \epsilon$, set $F \leftarrow F\,|\,l\,|\,\{l' \mid F\,|\,l \vdash_1 l'\}$ and return to E2. Otherwise if $F\,|\,\bar{l} \not\vdash_1 \epsilon$, $F \leftarrow F\,|\,\bar{l}\,|\,\{l' \mid F\,|\,\bar{l} \vdash_1 l'\}$ and return to E2. Otherwise terminate ('maybe'). ▌

Notice that this algorithm doesn't backtrack after committing itself in E2 to either l or $\bar{l}$.

a) If F consists of Horn clauses, possibly renamed (see exercise 7.1.1–55), prove that SLUR will never return 'maybe', regardless of how it chooses l in step E2.
b) Find four clauses F on three variables such that SLUR always returns 'sat', although F is *not* a set of possibly renamed Horn clauses.
c) Prove that SLUR never returns 'maybe' if and only if $F \in \mathrm{UC}_1$ (see exercise 443).
d) Explain how to implement SLUR in linear time with respect to total clause length.

▸ **445.** [*22*] Find short certificates of unsatisfiability for the pigeonhole clauses (106)–(107), when they are supplemented by (a) (181); (b) (182); (c) (183).

446. [*M10*] What's the maximum number of edges in a subgraph of $K_{m,n}$ that has girth ≥ 6? (Express your answer in terms of $Z(m,n)$.)

▸ **447.** [*22*] Determine the maximum number of edges in a girth-8 subgraph of $K_{8,8}$.

448. [*M25*] What is $Z(m,n)$ when m is odd and $n = m(m-1)/6$? *Hint:* See 6.5–(16).

449. [*21*] Exhibit $n \times n$ quad-free matrices that contain the maximum number of 1s and obey the lexicographic constraints (185), (186), for $8 \leq n \leq 16$.

450. [*25*] Prove that there is essentially only one 10×10 quad-free system of points and lines with 34 incidences. *Hint:* First show that every line must contain either 3 points or 4 points; hence every point must belong to either 3 lines or 4 lines.

▸ **451.** [*28*] Find a way to color the squares of a 10×10 board with three colors, so that no rectangle has four corners of the same color. Prove furthermore that every such "nonchromatic rectangle" board has the color distribution $\{34, 34, 32\}$, not $\{34, 33, 33\}$. But show that if any square of the board is removed, a nonchromatic rectangle is possible with 33 squares of each color.

hierarchy of hardness
UC_k
partial assignment
propagation completeness, see UC_1
PC_k
single lookahead unit resolution
SLUR
lookahead
backtrack
Horn clauses
renamed
certificates of unsatisfiability
pigeonhole
$K_{m,n}$
girth
$Z(m,n)$
nonchromatic rectangle

452. [*34*] Find a nonchromatic rectangle with *four* colors on an 18×18 board.

453. [*M23*] An $m \times n$ matrix $X = (x_{ij})$ is said to be *decomposable* if it has row indices $R \subseteq \{1, \ldots, m\}$ and column indices $C \subseteq \{1, \ldots, n\}$ such that $0 < |R| + |C| < m + n$, with $x_{ij} = 0$ whenever ($i \in R$ and $j \notin C$) or ($i \notin R$ and $j \in C$). It represents a bipartite graph on the vertices $\{u_1, \ldots, u_m\}$ and $\{v_1, \ldots, v_n\}$, if $[u_i — v_j] = [x_{ij} \neq 0]$.

a) Prove that X is indecomposable if and only if its bipartite graph is connected.

b) The *direct sum* $X' \oplus X''$ of matrices X' and X'', where X' is $m' \times n'$ and X'' is $m'' \times n''$, is the $(m' + m'') \times (n' + n'')$ "block diagonal" matrix X that has X' in its upper left corner, X'' in the lower right corner, and zeros elsewhere (see 7–(40)). True or false: If the rows and columns of X' and X'' are nonnegative and lexicographically ordered as in (185) and (186), so are the rows and columns of X.

c) Let X be any nonnegative matrix whose rows and columns are lexicographically nonincreasing, as in (185) and (186). True or false: X is decomposable if and only if X is a direct sum of smaller matrices X' and X''.

454. [*15*] If τ is an endomorphism for the solutions of f, show that $f(x) = f(x\tau)$ for every cyclic element x (every element that's in a cycle of τ).

455. [*M20*] Suppose we know that (187) is an endomorphism of some given clauses F on the variables $\{x_1, x_2, x_3, x_4\}$. Can we be sure that F is satisfiable if and only if $F \wedge C$ is satisfiable, when (a) $C = \bar{1}2\bar{4}$, i.e., $C = (\bar{x}_1 \vee x_2 \vee \bar{x}_4)$? (b) $C = 2\bar{3}\bar{4}$? (c) $C = 123$? (d) $C = 1\bar{3}4$?

456. [*M21*] For how many functions $f(x_1, x_2, x_3, x_4)$ is (187) an endomorphism?

457. [*HM19*] Show that every Boolean $f(x_1, x_2, x_3, x_4)$ has more than 51 quadrillion endomorphisms, and an n-variable function has more than $2^{2^n(n-1)}$.

458. [*20*] The simplification of clauses by removing an autarky can be regarded as the exploitation of an endomorphism. Explain why.

▸ **459.** [*20*] Let X_{ij} denote the submatrix of X consisting of the first i rows and the first j columns. Show that the numbers $\mathrm{sweep}(X_{ij})$ satisfy a simple recurrence, from which it's easy to compute $\mathrm{sweep}(X) = \mathrm{sweep}(X_{mn})$.

460. [*21*] Given m, n, k, and r, construct clauses that are satisfied by an $m \times n$ binary matrix $X = (x_{ij})$ if and only if $\mathrm{sweep}(X) \le k$ and $\sum_{i,j} x_{ij} \ge r$.

461. [*20*] What additional clauses will rule out non-fixed points of τ_1 and τ_2?

462. [*M22*] Explain why τ_1, τ_2, and τ_3 preserve satisfiability in the sweep problem.

▸ **463.** [*M21*] Show that X is a fixed point of τ_1, τ_2, and τ_3 if and only if its rows and columns are nondecreasing. Therefore the maximum of $\nu X = \sum_{i,j} x_{ij}$ over all binary matrices of sweep k is a simple function of m, n, and k.

▸ **464.** [*M25*] Transformations τ_1 and τ_2 don't change the text's example 10×10 matrix. Prove that they will never change *any* 10×10 matrix of sweep 3 that has $\nu X = 51$.

465. [*M21*] Justify the text's rule for simultaneous endomorphisms in the perfect matching problem: Any perfect matching must lead to one that's fixed by every τ_{uv}.

466. [*M23*] Prove that when mn is even, the text's even-odd rule (190) for endomorphisms of $m \times n$ domino coverings has exactly one fixed point.

467. [*20*] Mutilate the 7×8 and 8×7 boards by removing the upper right and lower left cells. What domino coverings are fixed by all the even-odd endomorphisms like (190)?

decomposable
bipartite graph
indecomposable
connected
direct sum
notation $A \oplus B$
block diagonal
lexicographically
endomorphism
autarky
submatrix
sweep
recurrence
perfect matching
domino coverings
fixed point
Mutilate
even-odd endomorphisms

468. [*20*] Experiment with the mutilated chessboard problem when the even-odd endomorphisms are modified so that (a) they use the *same* rule for all i and j; or (b) they each make an independent random choice between horizontal and vertical.

▸ **469.** [*M25*] Find a certificate of unsatisfiability $(C_1, C_2, \ldots, C_t)$ for the fact that an 8×8 chessboard minus cells $(1,8)$ and $(8,1)$ cannot be exactly covered by dominoes h_{ij} and v_{ij} that are fixed under all of the even-odd endomorphisms. Each C_k for $1 \le k < t$ should be a single positive literal. (Therefore the clauses for this problem belong to the relatively simple class PC_2 in the hierarchy of exercise 443.)

▸ **470.** [*M22*] Another class of endomorphisms, one for *every* 4-cycle, can also be used in perfect matching problems: Let the *vertices* (instead of the edges) be totally ordered in some fashion. Every 4-cycle can be written v_0 — v_1 — v_2 — v_3 — v_0, with $v_0 > v_1 > v_3$ and $v_0 > v_2$; the corresponding endomorphism changes any solution for which $v_0v_1 = v_2v_3 = 1$ by setting $v_0v_1 \leftarrow v_2v_3 \leftarrow 0$ and $v_1v_2 \leftarrow v_3v_0 \leftarrow 1$. Prove that every perfect matching leads to a fixed point of all these transformations.

471. [*16*] Find all fixed points of the mappings in exercise 470 when the graph is K_{2n}.

472. [*M25*] Prove that even-odd endomorphisms such as (190) in the domino covering problem can be regarded as instances of the endomorphisms in exercise 470.

▸ **473.** [*M23*] Generalize exercise 470 to endomorphisms for the unsatisfiable clauses of Tseytin's graph parity problems in exercise 245.

474. [*M20*] A signed permutation is a symmetry of $f(x)$ if and only if $f(x) = f(x\sigma)$ for all x, and it is an *antisymmetry* if and only if we have $f(x) = \bar{f}(x\sigma)$ for all x.

a) How many signed permutations of n elements are possible?
b) Write $75\bar{1}\bar{4}\bar{2}6\bar{3}$ in cycle form, as an unsigned permutation of $\{1, \ldots, 7, \bar{1}, \ldots, \bar{7}\}$.
c) For how many functions f of four variables is $\bar{4}13\bar{2}$ a symmetry?
d) For how many functions f of four variables is $\bar{4}13\bar{2}$ an antisymmetry?
e) For how many $f(x_1, \ldots, x_7)$ is $75\bar{1}\bar{4}\bar{2}6\bar{3}$ a symmetry or antisymmetry?

475. [*M22*] Continuing exercise 474, a Boolean function is called *asymmetric* if the identity is its only symmetry; it is *totally asymmetric* if it is asymmetric and has no antisymmetries.

a) If f is totally asymmetric, how many functions are equivalent to f under the operations of permuting variables, complementing variables, and/or complementing the function?
b) According to (a) and 7.1.1–(95), the function $(x \vee y) \wedge (x \oplus z)$ is not totally asymmetric. What is its nontrivial symmetry?
c) Prove that if f is not asymmetric, it has an automorphism of prime order p.
d) Show that if $(uvw)(\bar{u}\bar{v}\bar{w})$ is a symmetry of f, so is $(uv)(\bar{u}\bar{v})$.
e) Make a similar statement if f has a symmetry of the form $(uvwxy)(\bar{u}\bar{v}\bar{w}\bar{x}\bar{y})$.
f) Conclude that, if $n \le 5$, the Boolean function $f(x_1, \ldots, x_n)$ is totally asymmetric if and only if no signed involution is a symmetry or antisymmetry of f.
g) However, exhibit a counterexample to that statement when $n = 6$.

476. [*M23*] For $n \le 5$, find Boolean functions of n variables that are (a) asymmetric but not totally asymmetric; (b) totally asymmetric. Furthermore, your functions should be the easiest to evaluate (in the sense of having a smallest possible Boolean chain), among all functions that qualify. *Hint:* Combine exercises 475 and 477.

▸ **477.** [*23*] (*Optimum Boolean evaluation.*) Construct clauses that are satisfiable if and only if there is an r-step normal Boolean chain that computes m given functions g_1,

certificate of unsatisfiability
PC_2
hierarchy
4-cycle
complete graph
Tseytin
graph-based axioms
parity-related clauses
signed permutation
antisymmetry
cycle form
asymmetric
equivalence of Boolean functions
evaluation of Boolean functions
Boolean chain

..., g_m on n variables. (For example, if $n = 3$ and $g_1 = \langle x_1x_2x_3\rangle$, $g_2 = x_1 \oplus x_2 \oplus x_3$, such clauses with $r = 4$ and 5 enable a SAT solver to discover a "full adder" of minimum cost; see 7.1.2–(1) and 7.1.2–(22).) *Hint:* Represent each bit of the truth tables.

▸ **478.** [*23*] Suggest ways to break symmetry in the clauses of exercise 477.

▸ **479.** [*25*] Use SAT technology to find optimum circuits for the following problems:

a) Compute z_2, z_1, and z_0, when $x_1 + x_2 + x_3 + x_4 = (z_2z_1z_0)_2$ (see 7.1.2–(27)).
b) Compute z_2, z_1, and z_0, when $x_1 + x_2 + x_3 + x_4 + x_5 = (z_2z_1z_0)_2$.
c) Compute all four symmetric functions S_0, S_1, S_2, S_3 of $\{x_1, x_2, x_3\}$.
d) Compute all five symmetric functions S_0, S_1, S_2, S_3, S_4 of $\{x_1, x_2, x_3, x_4\}$.
e) Compute the symmetric function $S_3(x_1, x_2, x_3, x_4, x_5, x_6)$.
f) Compute the symmetric function $S_{0,4}(x_1, \ldots, x_6) = \big[(x_1 + \cdots + x_6) \bmod 4 = 0\big]$.
g) Compute all eight minterms of $\{x_1, x_2, x_3\}$ (see 7.1.2–(30)).

480. [*25*] Suppose the values 0, 1, 2 are encoded by the two-bit codes $x_lx_r = 00$, 01, and $1{*}$, respectively, where 10 and 11 both represent 2. (See Eq. 7.1.3–(120).)

a) Find an optimum circuit for mod 3 addition: $z_lz_r = (x_lx_r + y_ly_r) \bmod 3$.
b) Find an optimum circuit that computes $z_lz_r = (x_1 + x_2 + x_3 + y_ly_r) \bmod 3$.
c) Conclude that $\big[x_1 + \cdots + x_n \equiv a \pmod{3}\big]$ can be computed in $< 3n$ steps.

▸ **481.** [*28*] An ordered bit pair xy can be encoded by another ordered bit pair $[\![xy]\!] = (x \oplus y)y$ without loss of information, because $[\![xy]\!] = uv$ implies $[\![uv]\!] = xy$.

a) Find an optimum circuit that computes $([\![zz']\!])_2 = x_1 + x_2 + x_3$.
b) Let $\nu[\![uv]\!] = (u \oplus v) + v$, and note that $\nu[\![00]\!] = 0$, $\nu[\![01]\!] = 2$, $\nu[\![1{*}]\!] = 1$. Find an optimum circuit that, given $x_1 \ldots x_5$, computes $z_1z_2z_3$ such that we have $\nu[\![x_1x_2]\!] + \nu[\![x_3x_4]\!] + x_5 = 2\nu[\![z_1z_2]\!] + z_3$.
c) Use that circuit to prove by induction that the "sideways sum" $(z_{\lfloor \lg n\rfloor} \ldots z_1z_0)_2 = x_1 + x_2 + \cdots + x_n$ can always be computed with fewer than $4.5n$ gates.

▸ **482.** [*26*] (*Erdős discrepancy patterns.*) The binary sequence $y_1 \ldots y_t$ is called *strongly balanced* if we have $\big|\sum_{j=1}^{k}(2y_j - 1)\big| \le 2$ for $1 \le k \le t$.

a) Show that this balance condition needs to be checked only for odd $k \ge 3$.
b) Describe clauses that efficiently characterize a strongly balanced sequence.
c) Construct clauses that are satisfied by $x_1x_2 \ldots x_n$ if and only if $x_dx_{2d} \ldots x_{\lfloor n/d\rfloor d}$ is strongly balanced for $1 \le d \le n$.

483. [*21*] Symmetry between colors was broken in the coloring problems of Table 6 by assigning fixed colors to a large clique in each graph. But many graphs have no large clique, so a different strategy is necessary. Explain how to encode the "restricted growth string" principle (see Section 7.2.1.5) with appropriate clauses, given an ordering $v_1v_2 \ldots v_n$ of the vertices: The color of v_j must be at most one greater than the largest color assigned to $\{v_1, \ldots, v_{j-1}\}$. (In particular, v_1 always has color 1.)

Experiment with this scheme by applying it to the *book* graphs `anna`, `david`, `homer`, `huck`, and `jean` of the Stanford GraphBase.

484. [*22*] (*Graph quenching.*) A graph with vertices $(v_1, \ldots, v_n)$ is called "quenchable" if either (i) $n = 1$; or (ii) there's a k such that v_k — v_{k+1} and the graph on $(v_1, \ldots, v_{k-1}, v_{k+1}, \ldots, v_n)$ can be quenched; or (iii) there's an l such that v_l — v_{l+3} and the graph on $(v_1, \ldots, v_{l-1}, v_{l+3}, v_{l+1}, v_{l+2}, v_{l+4}, \ldots, v_n)$ can be quenched.

a) Find a 4-element graph that is quenchable although $v_3 \not\!\!—\, v_4$.
b) Construct clauses that are satisfiable if and only if a given graph is quenchable. *Hint:* Use the following three kinds of variables for this model-checking problem:

full adder
truth tables
break symmetry
symmetric functions
Mod 4 parity
minterms
binary decoder
encoding of ternary data
representing three states with two bits
mapping three items into two-bit codes
Mod 3 parity
sideways sum
Erdős discrepancy patterns
strongly balanced
Symmetry between colors
coloring problems
clique
encode
restricted growth string
book graphs
Stanford GraphBase
Graph quenching
quenchable
model-checking problem

$x_{t,i,j} = [v_i \!-\!\!-\! v_j$ at time $t]$, for $1 \le i < j \le n-t$; $q_{t,k} =$ [a quenching move of type (ii) leads to time $t+1$]; $s_{t,l} =$ [a quenching move of type (iii) leads to time $t+1$].

▸ **485.** [*23*] Sometimes successive transitions in the previous exercise are commutative: For example, the effect of $q_{t,k}$ and $q_{t+1,k+1}$ is the same as $q_{t,k+2}$ and $q_{t+1,k}$. Explain how to break symmetry in such cases, by allowing only one of the two possibilities.

486. [*21*] (*Late Binding solitaire.*) Shuffle a deck and deal out 18 cards; then try to reduce these 18 piles to a single pile, using a sequence of "captures" in which one pile is placed on top of another pile. A pile can capture only the pile to its immediate left, or the pile found by skipping left over two other piles. Furthermore a capture is permitted only if the top card in the capturing pile has the same suit or the same rank as the top card in the captured pile. For example, consider the following deal:

J♡ 5♡ 10♣ 8♢ J♣ A♣ K♠ A♡ 4♣ 8♠ 5♠ 5♢ 2♢ 10♠ A♠ 6♡ 3♡ 10♢

Ten captures are initially possible, including 5♡ × J♡, A♣ × ×10♣, and 5♢ × 5♠. Some captures then make others possible, as in 8♠ × × K♠ × × 8♢.

If captures must be made "greedily" from left to right as soon as possible, this game is the same as the first 18 steps of a classic one-player game called "Idle Year," and we wind up with five piles [see *Dick's Games of Patience* (1883), 50–52]. But if we cleverly hold back until all 18 cards have been dealt, we can do much better.

Show that one can win from this position, but not if the first move is A♣ × J♣.

▸ **487.** [*27*] There are $\binom{64}{8} = 4426165368$ ways to place eight queens on a chessboard. Long ago, W. H. Turton asked which of them causes the maximum number of vacant squares to remain unattacked. [See W. W. Rouse Ball, *Mathematical Recreations and Problems*, third edition (London: Macmillan, 1896), 109–110.]

Every subset S of the vertices of a graph has three *boundary sets* defined thus:

$$\partial S = \text{the set of all edges with exactly one endpoint} \in S;$$
$$\partial_{\rm out} S = \text{the set of all vertices} \notin S \text{ with at least one neighbor} \in S;$$
$$\partial_{\rm in} S = \text{the set of all vertices} \in S \text{ with at least one neighbor} \notin S.$$

Find the minimum and maximum sizes of ∂S, $\partial_{\rm out} S$, and $\partial_{\rm in} S$, over all 8-element sets S in the queen graph Q_8 (exercise 7.1.4–241). Which set answers Turton's question?

▸ **488.** [*24*] (*Peaceable armies of queens.*) Prove that armies of nine white queens and nine black queens can coexist on a chessboard without attacking each other, but armies of size 10 cannot, by devising appropriate sets of clauses and applying Algorithm C. Also examine the effects of symmetry breaking. (This problem has sixteen symmetries, because we can swap colors and/or rotate and/or reflect the board.) How large can coexisting armies of queens be on $n \times n$ boards, for $n \le 11$?

489. [*M21*] Find a recurrence for T_n, the number of signed involutions on n elements.

▸ **490.** [*15*] Does Theorem E hold also when $p_1 p_2 \ldots p_n$ is any *signed* permutation?

▸ **491.** [*22*] The unsatisfiable clauses R in (6) have the signed permutation $234\bar{1}$ as an automorphism. How can this fact help us to verify their unsatisfiability?

492. [*M20*] Let τ be a *signed mapping* of the variables $\{x_1, \ldots, x_n\}$; for example, the signed mapping '$\bar{4}13\bar{3}$' stands for the operation $(x_1, x_2, x_3, x_4) \mapsto (x_{\bar{4}}, x_1, x_3, x_{\bar{3}}) = (\bar{x}_4, x_1, x_3, \bar{x}_3)$. When a signed mapping is applied to a clause, some of the resulting literals might coincide; or two literals might become complementary, making a tautology. When $\tau = \bar{4}13\bar{3}$, for instance, we have $(123)\tau = \bar{4}13$, $(13\bar{4})\tau = \bar{4}3$, $(1\bar{3}\bar{4})\tau = \wp$.

commutative
break symmetry
Late Binding solitaire
solitaire
patience
playing cards
Idle Year
Dick
queens
chessboard
Turton
Ball
boundary sets
queen graph
armies of queens
queens
symmetry breaking
coexisting armies of queens
signed involutions
signed
automorphism
signed mapping
tautology

A family F of clauses is said to be “closed” under a signed mapping τ if $C\tau$ is subsumed by some clause of F whenever $C \in F$. Prove that τ is an endomorphism of F in such a case.

493. [*20*] The problem *waerden*$(3, 3; 9)$ has four symmetries, because we can reflect and/or complement all the variables. How can we speed up the proof of unsatisfiability by adding clauses to break those symmetries?

494. [*21*] Show that if $(uvw)(\bar u\bar v\bar w)$ is a symmetry of some clauses F, we’re allowed to break symmetries as if $(uv)(\bar u\bar v)$, $(uw)(\bar u\bar w)$, and $(vw)(\bar v\bar w)$ were also symmetries. For example, if $i < j < k$ and if $(ijk)(\bar i\bar j\bar k)$ is a symmetry, we can assert $(\bar x_i \vee x_j) \wedge (\bar x_j \vee x_k)$ with respect to the global ordering $p_1 \ldots p_n = 1 \ldots n$. What are the corresponding binary clauses when the symmetry is (i) $(ij\bar k)(\bar i\bar jk)$? (ii) $(i\bar jk)(\bar ij\bar k)$? (iii) $(i\bar j\bar k)(\bar i\bar jk)$?

495. [*M22*] Spell out the details of how we can justify appending clauses to assert (185) and (186), using Corollary E, whenever we have an $m \times n$ problem whose variables x_{ij} possess both row and column symmetry. (In other words we assume that $x_{ij} \mapsto x_{(i\pi)(j\rho)}$ is an automorphism for all permutations π of $\{1, \ldots, m\}$ and ρ of $\{1, \ldots, n\}$.)

▸ **496.** [*M20*] B. C. Dull reasoned as follows: “The pigeonhole clauses have row and column symmetry. Therefore we can assume that the rows are lexicographically increasing from top to bottom, and the columns are lexicographically increasing from right to left. Consequently the problem is easily seen to be unsatisfiable.” Was he correct?

497. [*22*] Use BDD methods to determine the number of 8×8 binary matrices that have both rows and columns in nondecreasing lexicographic order. How many of them have exactly r 1s, for $r = 24$, $r = 25$, $r = 64 - 25 = 39$, and $r = 64 - 24 = 40$?

498. [*22*] Justify adding the symmetry-breakers (183) to the pigeonhole clauses.

499. [*21*] In the pigeonhole problem, is it legitimate to include the clauses (183) together with clauses that enforce lexicographic row and column order?

500. [*16*] The precocious student J. H. Quick decided to extend the monkey wrench principle, arguing that if $F_0 \cup S \vdash l$ then the original clauses F can be replaced by $F \,|\, l$. But he soon realized his mistake. What was it?

501. [*22*] Martin Gardner introduced an interesting queen placement problem in *Scientific American* **235**, 4 (October 1976), 134–137: “Place r queens on an $m \times n$ chessboard so that (i) no three are in the same row, column, or diagonal; (ii) no empty square can be occupied without breaking rule (i); and (iii) r is as small as possible.” Construct clauses that are satisfiable if and only if there’s a solution to conditions (i) and (ii) with at most r queens. (A similar problem was considered in exercise 7.1.4–242.)

502. [*16*] (*Closest strings.*) Given binary strings $s_1, \ldots, s_m$ of length n, and threshold parameters $r_1, \ldots, r_m$, construct clauses that are satisfiable by $x = x_1 \ldots x_n$ if and only if x differs from s_j in at most r_j positions, for $1 \le j \le m$.

503. [*M20*] (*Covering strings.*) Given s_j and r_j as in exercise 502, show that *every* string of length n is within r_j bits of *some* s_j if and only if the closest string problem has no solution with parameters $r'_j = n - 1 - r_j$.

▸ **504.** [*M21*] The problem in exercise 502 can be proved NP-complete as follows:

a) Let w_j be the string of length $2n$ that is entirely 0 except for 1s in positions $2j-1$ and $2j$, and let $w_{n+j} = \bar w_j$, for $1 \le j \le n$. Describe all binary strings of length $2n$ that differ from each of $w_1, \ldots, w_{2n}$ in at most n bit positions.

b) Given a clause $(l_1 \vee l_2 \vee l_3)$ with strictly distinct literals $l_1, l_2, l_3 \in \{x_1, \ldots, x_n, \bar x_1, \ldots, \bar x_n\}$, let y be the string of length $2n$ that is entirely zero except that it has

subsumed
waerden
Dull
pigeonhole
BDD
lexicographic order
Quick
monkey wrench principle
Gardner
queen placement
Closest strings
binary strings
noisy data
Covering strings
NP-complete

1 in position $2k-1$ when some l_i is $\bar{x}_k$, and 1 in position $2k$ when some l_i is x_k. In how many bit positions does a string that satisfies (a) differ from y?

c) Given a 3SAT problem F with m clauses and n variables, use (a) and (b) to construct strings $s_1, \ldots, s_{m+2n}$ of length $2n$ such that F is satisfiable if and only if the closest string problem is satisfiable with $r_j = n + [j > 2n]$.

d) Illustrate your construction in (c) by exhibiting the closest string problems that correspond to the simple 3SAT problems R and R' in (6) and (7).

505. [*21*] Experiment with making Algorithm L nondeterministic, by randomizing the initial order of `VAR` in step L1 just as `HEAP` is initialized randomly in step C1. How does the modified algorithm perform on, say, problems D3, K0, and W2 of Table 6?

506. [*22*] The *weighted variable interaction graph* of a family of clauses has one vertex for each variable and the weight $\sum 2/(|c|(|c|-1))$ between vertices u and v, where the sum is over all clauses c that contain both $\pm u$ and $\pm v$. Figure 52 indicates these weights indirectly, by making the heavier edges darker.

a) True or false: The sum of all edge weights is the total number of clauses.

b) Explain why the graph for test case B2 has exactly 6 edges of weight 2. What are the weights of the other edges in that graph?

▸ **507.** [*21*] (Marijn Heule.) Explain why "windfalls" (see (72)) help Algorithm L to deal with miter problems such as D5.

508. [*M20*] According to Table 7, Algorithm C proved problem T3 to be unsatisfiable after learning about 323 thousand clauses. About how many times did it enter a purging phase in step C7?

509. [*20*] Several of the "training set" tasks used when tuning Algorithm C's parameters were taken from the 100 test cases of Table 6. Why didn't this lead to a problem of "overfitting" (namely, of choosing parameters that are too closely associated with the trainees)?

510. [*18*] When the data points A1, A2, ..., X8 were plotted in Fig. 55, one by one, they sometimes covered parts of previously plotted points, because of overlaps. What test cases are partially hidden by (a) T2? (b) X6? (c) X7?

511. [*22*] Problem P4 in Table 6 is a strange set of clauses that lead to extreme behavior of Algorithm C in Figs. 54 and 55; and it causes Algorithm L to "time out" in Fig. 53.

a) The preprocessing algorithm of the text needs about 1.5 megamems to convert those 2509 clauses in 400 variables into just 2414 clauses in 339 variables. Show empirically that Algorithm L makes short work of the resulting 2414 clauses.

b) How efficient is Algorithm C on those preprocessed clauses?

c) What is the behavior of WalkSAT on P4, with and without preprocessing?

512. [*29*] Find parameters for Algorithm C that will find an Erdős discrepancy pattern $x_1x_2\ldots x_n$ rapidly when $n = 500$. (This is problem E0 in Table 6.) Then compare the running times of nine random runs with your parameters versus nine random runs with (194), when $n = 400, 500, 600, \ldots, 1100, 1160$, and 1161.

513. [*24*] Find parameters for Algorithm L that tune it for $rand(3, m, n, seed)$.

514. [*24*] The timings quoted in the text for Algorithm W, for problems in Table 6, are based on the median of nine runs using the parameters $p = .4$ and $N = 50n$, restarting from scratch if necessary until a solution is found. Those parameters worked fine in most cases, unless Algorithm W was unsuited to the task. But problem C9 was solved more quickly with $p = .6$ and $N = 2500n$ (943 Mμ versus 9.1 Gμ).

Find values of p and N/n that give near-optimum performance for problem C9.

3SAT
closest string
Rivest's clauses
nondeterministic
randomizing
variable interaction graph
Heule
windfalls
miter problems
purging
training set
tuning
overfitting
preprocessing
lookahead solver versus conflict driven
WalkSAT
Erdős discrepancy pattern
discrepancy
rand
WalkSAT

▶ **515.** [*23*] (*Hard sudoku.*) Specify SAT clauses with which a designer of sudoku puzzles can meet the following specifications: (i) If cell (i,j) of the puzzle is blank, so is cell $(10-i, 10-j)$, for $1 \le i,j \le 9$. (ii) Every row, every column, and every box contains at least one blank. (Here "box" means one of sudoku's nine special 3×3 subarrays.) (iii) No box contains an all-blank row or an all-blank column. (iv) There are at least two ways to fill every blank cell, without conflicting with nonblank entries in the same row, column, or box. (v) If a row, column, or box doesn't already contain k, there are at least two places to put k into that row, column, or box, without conflict. (vi) If the solution has a 2×2 subarray of the form $\substack{k\,l\\l\,k}$, those four cells must not all be blank.

(Condition (i) is a feature of "classic" sudoku puzzles. Conditions (iv) and (v) ensure that the corresponding exact cover problem has no forced moves; see Section 7.2.2.1. Condition (vi) rules out common cases with non-unique solutions.)

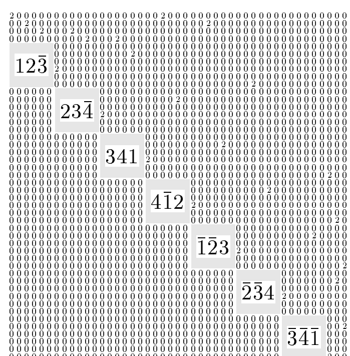
516. [*M49*] Prove or disprove the *strong exponential time hypothesis*: "If $\tau < 2$, there is an integer k such that no randomized algorithm can solve every kSAT problem in fewer than τ^n steps, where n is the number of variables."

517. [*25*] Given clauses $C_1, \ldots, C_m$, the *one-per-clause* satisfiability problem asks if there is a Boolean assignment $x_1 \ldots x_n$ such that every clause is satisfied by a *unique* literal. In other words, we want to solve the simultaneous equations $\Sigma C_j = 1$ for $1 \le j \le m$, where ΣC is the sum of the literals of clause C.

a) Prove that this problem is NP-complete, by reducing 3SAT to it.

b) Prove that this problem, in turn, can be reduced to its special case "*one-in-three* satisfiability," where every given clause is required to be ternary.

518. [*M32*] Given a 3SAT problem with m clauses and n variables, we shall construct a $(6m+n) \times (6m+n)$ matrix M of integers such that the *permanent*, per M, is zero if and only if the clauses are unsatisfiable. For example, the solvable problem (7) corresponds to the 46×46 matrix indicated here; each shaded box stands for a fixed 6×6 matrix A that corresponds to a clause.



Each A has three "inputs" in columns 1, 3, 5 and three "outputs" in rows 2, 4, 6. The first n rows and the last n columns correspond to variables. Outside of the As, all entries are either 0 or 2; and the 2s link variables to clauses, according to a scheme much like the data structures in several of the algorithms in this section: Let I_{ij} and O_{ij} denote the jth input and output of clause i, for $1 \le i \le m$ and $1 \le j \le 3$. Then, if literal l appears in $t \ge 0$ clauses $i_1 < \cdots < i_t$, as element $j_1, \ldots, j_t$, we put '2' in column $I_{i_{k+1}j_{k+1}}$ of row $O_{i_k j_k}$ for $0 \le k \le t$ ($O_{i_0 j}$ is row $|l|$, $I_{i_{t+1}j}$ is column $6m+|l|$).

a) Find a 6×6 matrix $A = (a_{ij})$, whose elements are either 0, 1, or -1, such that

$$\operatorname{per}\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} \\ a_{21}+2r & a_{22} & a_{23}+2s & a_{24} & a_{25}+2t & a_{26} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} \\ a_{41}+2u & a_{42} & a_{43}+2v & a_{44} & a_{45}+2w & a_{46} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} \\ a_{61}+2x & a_{62} & a_{63}+2y & a_{64} & a_{65}+2z & a_{66} \end{pmatrix} = 16\left(\operatorname{per}\begin{pmatrix} r+1 & s & t \\ u & v+1 & w \\ x & y & z+1 \end{pmatrix} - 1\right).$$

Hint: There's a solution with lots of symmetry.

b) In which of the rows and columns of M does '2' occur twice? once? not at all?

c) Conclude that per $M = 2^{4m+n}s$, when the 3SAT problem has exactly s solutions.

Hard sudoku
sudoku
exact cover problem
strong exponential time hypothesis
kSAT
one-per-clause
NP-complete
3SAT
one-in-three
ternary
permanent
gadget

519. [*20*] Table 7 shows inconclusive results in a race for factoring between *factor_fifo* and *factor_lifo*. What is the comparable performance of *factor_rand*$(m, n, z, 314159)$?

▸ **520.** [*24*] Every instance of SAT corresponds in a natural way to an *integer programming feasility* problem: To find, if possible, integers $x_1, \ldots, x_n$ that satisfy the linear inequalities $0 \le x_j \le 1$ for $1 \le j \le n$ and

$$l_1 + l_2 + \cdots + l_k \ge 1 \qquad \text{for each clause } C = (l_1 \vee l_2 \vee \cdots \vee l_k).$$

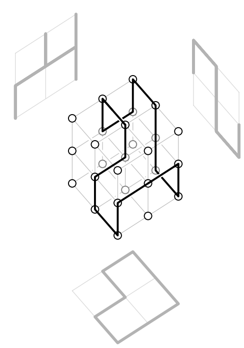
For example, the inequality that corresponds to the clause $(x_1 \vee \bar{x}_3 \vee \bar{x}_4 \vee x_7)$ is $x_1 + (1-x_3) + (1-x_4) + x_7 \ge 1$; i.e., $x_1 - x_3 - x_4 + x_7 \ge -1$.

Sophisticated "IP solvers" have been developed by numerous researchers for solving general systems of integer linear inequalities, based on techniques of "cutting planes" in high-dimensional geometry. Thus we can solve any satisfiability problem by using such general-purpose software, as an alternative to trying a SAT solver.

Study the performance of the best available IP solvers, with respect to the 100 sets of clauses in Table 6, and compare it to the performance of Algorithm C in Table 7.

521. [*30*] Experiment with the following idea, which is much simpler than the clause-purging method described in the text: "Forget a learned clause of length k with probability p_k," where $p_1 \ge p_2 \ge p_3 \ge \cdots$ is a tunable sequence of probabilities.

▸ **522.** [*26*] (*Loopless shadows.*) A cyclic path within the cube $P_3 \mathbin{\square} P_3 \mathbin{\square} P_3$ is shown here, together with the three "shadows" that appear when it is projected onto each coordinate plane. Notice that the shadow at the bottom contains a loop, but the other two shadows do not. Does this cube contain a cycle whose three shadows are entirely *without* loops? Use SAT technology to find out.



523. [*30*] Prove that, for any m or n, no cycle of the graph $P_m \mathbin{\square} P_n \mathbin{\square} P_2$ has loopless shadows.

▸ **524.** [*22*] Find all *Hamiltonian paths* of the cube $P_3 \mathbin{\square} P_3 \mathbin{\square} P_3$ that have loopless shadows.

▸ **525.** [*40*] Find the most difficult 3SAT problem you can that has at most 100 variables.

526. [*M25*] (David S. Johnson, 1974.) If F has m clauses, all of size $\ge k$, prove that some assignment leaves at most $m/2^k$ clauses unsatisfied.

999. [*M00*] this is a temporary exercise (for dummies)

factor_fifo
factor_lifo
factor_rand
integer programming
linear inequalities
IP solvers
cutting planes
100 sets of clauses
purging
Loopless shadows
shadows
projected
Hamiltonian paths
difficult
3SAT
Johnson
MAXSAT lower bound

SECTION 7.2.2.2

1. (a) $\emptyset$ (no clauses). (b) $\{\epsilon\}$ (one clause, which is empty).

2. Letting $1 \leftrightarrow$ lazy, $2 \leftrightarrow$ happy, $3 \leftrightarrow$ unhealthy, $4 \leftrightarrow$ dancer, we're given the respective clauses $\{314, \bar{1}42, 3\bar{4}2, \bar{2}4\bar{3}, \bar{1}3\bar{2}, 2\bar{3}1, \bar{1}4\bar{3}\}$, matching R' in (7). So all known Pincusians dance happily, and none are lazy. But we know nothing about their health. [And we might wonder why travelers have bothered to describe so many empty sets.]

3. $f(j-1,n) + f(k-1,n)$, where $f(p,n) = \sum_{d=1}^{q}(n-pd) = p\binom{q}{2} + q(n \bmod p) \approx n^2/(2p)$, if we set $q = \lfloor n/p \rfloor$.

4. Those constraints are unsatisfiable if and only if we remove a subset of either $\{357, 456, \bar{3}\bar{5}\bar{7}, \bar{4}\bar{5}\bar{6}\}$, $\{246, 468, \bar{2}\bar{4}\bar{6}, \bar{4}\bar{6}\bar{8}\}$, $\{246, 357, 468, \bar{4}\bar{5}\bar{6}\}$, or $\{456, \bar{2}\bar{4}\bar{6}, \bar{3}\bar{5}\bar{7}, \bar{4}\bar{6}\bar{8}\}$.

5. No polynomial upper bound for $W(3,k)$ is currently known. Clearly $W(3,k)$ is less than $\overline{W}(3,k)$, the minimum n that guarantees either three equally spaced 0s or k *consecutive* 1s. An analysis by R. L. Graham in *Integers* **6** (2006), A29:1–A29:5, beefed up by a subsequent theorem of T. H. Bloom in `arXiv:1405.5800` [math.NT] (2014), 22 pages, shows that $\overline{W}(3,k) = \exp O(k(\log k)^4)$.

6. Let each x_i be 0 with probability $p = (2\ln k)/k$, and let n be at most $k^2/(\ln k)^3$. There are two kinds of "bad events": A_i, a set of three equally spaced 0s, occurs with probability $P = p^3$; and A'_j, a set of k equally spaced 1s, occurs with probability $P' = (1-p)^k \le \exp(-kp) = 1/k^2$. In the lopsidependency graph, which is bipartite, each A_i is adjacent to at most $D = 3k^3/((k-1)(\ln k)^3)$ nodes A'_j; each A'_j is adjacent to at most $d = \frac{3}{2}k^3/(\ln k)^3$ nodes A_i. By Theorem J, we want to show that, for all sufficiently large values of k, $P \le y(1-x)^D$ and $P' \le x(1-y)^d$, for some x and y.

Choose x and y so that $(1-x)^D = 1/2$ and $y = 2P$. Then $x = \Theta((\log k)^3/k^2)$ and $y = \Theta((\log k)^3/k^3)$; hence $(1-y)^d = \exp(-yd + O(y^2 d)) = O(1)$. [See T. Brown, B. M. Landman, and A. Robertson, *J. Combinatorial Theory* **A115** (2008), 1304–1309.]

7. Yes, for all n, when $x_1x_2x_3\ldots = 001001001\ldots$.

8. For example, let $x_{i,a}$ signify that $x_i = a$, for $1 \le i \le n$ and $0 \le a < b$. The relevant clauses are then $x_{i,0} \lor \cdots \lor x_{i,b-1}$ for $1 \le i \le n$; and $\bar{x}_{i,a} \lor \bar{x}_{i+d,a} \lor \cdots \lor \bar{x}_{i+(k_a-1)d,a}$, for $1 \le i \le n-(k_a-1)d$ and $d \ge 1$. Optionally include the clauses $\bar{x}_{i,a} \lor \bar{x}_{i,a'}$ for $0 \le a < a' < b$. (Whenever the relevant clauses are satisfiable, we can also satisfy the optional ones by falsifying some variables if necessary.)

[V. Chvátal found $W(3,3,3) = 27$. Kouril's paper shows that $W(2,4,8) = 157$, $W(2,3,14) = 202$, $W(2,5,6) = 246$, $W(4,4,4) = 293$, and lists many smaller values.]

9. $W(2,2,k) = 3k - (2,0,2,2,1,0)$ when $k \bmod 6 = (0,1,2,3,4,5)$. The sequence $2^{k-1}02^{k-1}12^{k-1}$ is maximal when $k \perp 6$; also $2^{k-1}02^{k-1}12^{k-3}$ when $k \bmod 6 = 3$; also $2^{k-1}02^{k-2}12^{k-1}$ when $k \bmod 6 = 4$; otherwise $2^{k-1}02^{k-2}12^{k-2}$. [See B. Landman, A. Robertson, and C. Culver, *Integers* **5** (2005), A10:1–A10:11, where many other values of $W(2,\ldots,2,k)$ are also established.]

10. If the original variables are $\{1,\ldots,n\}$, let the new ones be $\{1,\ldots,n\} \cup \{1',\ldots,n'\}$. The new problem has positive clauses $\{11',\ldots,nn'\}$. Its negative clauses are, for example, $\bar{2}'\bar{6}\bar{7}\bar{9}'$ if $2\bar{6}\bar{7}9$ was an original clause. The original problem is equivalent because it can be obtained from the new one by resolving away the primed variables.

[One can in fact construct an equivalent monotonic problem of size $O(m+n)$ in which $(x_1 \lor \cdots \lor x_k)$ is a positive clause if and only if $(\bar{x}_1 \lor \cdots \lor \bar{x}_k)$ is a negative clause. Such a problem, "not-all-equal SAT," is equivalent to 2-colorability of hypergraphs. See L. Lovász, *Congressus Numerantium* **8** (1973), 3–12; H. Kleine Büning and T. Lettmann, *Propositional Logic* (Cambridge Univ. Press, 1999), §3.2, Problems 4–8.]

$\emptyset$
empty clause
nullary clause
empty sets
unsatisfiable core
Graham
Bloom
lopsidependency graph
Brown
Landman
Robertson
Chvátal
Kouril
Landman
Robertson
Culver
resolving
not-all-equal SAT
hypergraph 2-colorability
2-colorability of hypergraphs
Lovász
Kleine Büning
Lettmann

auxiliary variables
Langford's problem
exact cover problem
*non*primary columns
Heule
complete graph
pigeons
kernel
maximal planar graph

11. For each variable i, the only way to match vertices of the forms ij' and ij'' is to choose all of its true triples or all of its false triples.

For each clause j, the vertex pairs $\{j'2, j'3\}$, $\{j'4, j'5\}$, $\{j'6, j'7\}$ define three "slots"; hence two of the vertices $\{wj, xj, yj, zj\}$ must be matched into the same slot. Furthermore we can't have two in one slot and two in another, because the remaining slot would then be unmatched. Thus two of the $\bar{l}j$ vertices are matched in their slot, while the other is matched with $j'1$, whenever we have a perfect matching.

Conversely, if all clauses are satisfied, with l_k true in clause j, there always are exactly two ways to match $\bar{l}_kj$ with $j'1$ while matching wj, xj, yj, zj, and the other two $\bar{l}j$ vertices with $j'2$, $\ldots$, $j'7$. (It's a beautiful construction! Notice that no vertex appears in more than three triples.)

12. Equation (13) says $S_1(y_1, \ldots, y_p) = S_{\geq 1}(y_1, \ldots, y_p) \wedge S_{\leq 1}(y_1, \ldots, y_p)$. If $p \leq 4$, use $\bigwedge_{1\leq j<k\leq p}(\bar{y}_j \vee \bar{y}_k)$ for $S_{\leq 1}(y_1, \ldots, y_p)$; otherwise $S_{\leq 1}(y_1, \ldots, y_p)$ can be encoded recursively via the clauses $S_{\leq 1}(y_1, y_2, y_3, t) \wedge S_{\leq 1}(\bar{t}, y_4, \ldots, y_p)$, where t is a new variable. [This method saves half of the auxiliary variables in the answer to exercise 7.1.1–55(b).]

Note: Langford's problem involves primary columns only; in an exact cover problem with *non*primary columns, such columns only need the constraint $S_{\leq 1}(y_1, \ldots, y_p)$.

13. (a) $S_1(x_1, x_2, x_3, x_4, x_5, x_6) \wedge S_1(x_7, x_8, x_9, x_{10}, x_{11}) \wedge S_1(x_{12}, x_{13}) \wedge S_1(x_{14}, x_{15}, x_{16}) \wedge S_1(x_1, x_7, x_{12}, x_{14}) \wedge S_1(x_2, x_8, x_{13}, x_{15}) \wedge S_1(x_1, x_3, x_9, x_{16}) \wedge S_1(x_2, x_4, x_7, x_{10}) \wedge S_1(x_3, x_5, x_8, x_{11}, x_{12}) \wedge S_1(x_4, x_6, x_9, x_{13}, x_{14}) \wedge S_1(x_5, x_{10}, x_{15}) \wedge S_1(x_6, x_{11}, x_{16})$.

(b) Duplicate clauses occur when rows intersect more than once. We avoid them if we simply generate clauses $\bar{x}_i \vee \bar{x}_j$ for every pair (i, j) of intersecting rows.

(c) When *langford*(4) is generated in this way, it has 85 distinct clauses in 16 variables, namely $(x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5 \vee x_6) \wedge (x_7 \vee x_8 \vee x_9 \vee x_{10} \vee x_{11}) \wedge \cdots \wedge (x_6 \vee x_{11} \vee x_{16}) \wedge (\bar{x}_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee \bar{x}_3) \wedge \cdots \wedge (\bar{x}_{15} \vee \bar{x}_{16})$.

But *langford*$'$(4) cannot use the trick of (b). It has 85 (nondistinct) clauses in 20 variables, beginning with 123456, $\bar{1}\bar{2}$, $\bar{1}\bar{3}$, $\bar{1}\bar{1}'$, $\bar{2}\bar{3}$, $\bar{2}\bar{1}'$, $\bar{3}\bar{1}'$, $1'\bar{4}$, $1'\bar{5}$, $1'\bar{6}$, $\bar{4}\bar{5}$, $\bar{4}\bar{6}$, $\bar{5}\bar{6}$, $\ldots$, if we denote the auxiliary variables by $1'$, $2'$, $\ldots$. Two of those clauses ($\bar{1}\bar{3}$ and $\bar{4}\bar{6}$) are repeated. (Incidentally, *langford*$'$(12) has 1548 clauses, 417 variables, 3600 cells.)

14. (Answer by M. Heule.) Those clauses sometimes help to focus the search. For example, if we're trying to color the complete graph K_n with n colors (or pigeons), we don't want to waste time trying $v_2 = 1$ when v_1 is already 1.

On the other hand, other instances of SAT often run slower when redundant clauses are present, because more updates to the data structures are needed.

We might also take an opposite approach, and replace (17) by nd clauses that force every color class to be a *kernel*. (See exercise 21.) Such clauses sometimes speed up a proof of uncolorability.

15. There are $N = n(n+1)$ vertices (j, k) for $0 \leq j \leq n$ and $0 \leq k < n$. If $(j, k) = (1, 0)$ we define $(j, k) — (n, i)$ for $x \leq i < n$, where $x = \lfloor n/2 \rfloor$. Otherwise we define the following edges: $(j, k) — (j+1, k+1)$ if $j < n$ and $k < n-1$; $(j, k) — (j+1, k)$ if $j < n$ and $j \neq k$; $(j, k) — (j, k+1)$ if $k < n-1$ and $j \neq k+1$; $(j, k) — (n, n-1)$ if $j = 0$; $(j, k) — (n-j, 0)$ if $k < n-1$ and $j = k$; $(j, k) — (n+1-j, 0)$ if $j > 0$ and $j = k$; $(j, k) — (n-j, n-j-1)$ if $k = n-1$ and $0 < j < k$; $(j, k) — (n+1-j, n-j)$ if $k = n-1$ and $0 < j < n$. Finally, $(0, 0) — (1, 0)$, and $(0, 0) — (n, i)$ for $1 \leq i \leq x$. That makes a grand total of $3N - 6$ edges (as it should in a maximal planar graph, according to exercise 7–46).

16. There's a unique 4-clique when $n \geq 5$, namely $\{(0, n-2), (0, n-1), (1, n-1), (n, n-1)\}$. All other vertices, except $(0, 0)$ and $(1, 0)$, are surrounded by neighbors

that form an induced cycle of length 4 or more (usually 6). [See J.-L. Laurier, *Artificial Intelligence* **14** (1978), 117.]

17. Let *mcgregor*(n) be the clauses (15) and (16) for the graph. Add clauses (19), for symmetric threshold functions to bound the number of variables v_1 for color 1; the kth vertex x_k can be specified by the ordering in answer 20. Then if, for instance, we can satisfy those clauses together with the unit clause s_r^N, where $N = n(n+1)$, we have proved that $f(n) < r$. Similarly, if we can satisfy them together with $\bar{s}_r^N$, we have proved that $g(n) \geq r$. Additional unit clauses that specify the colors of the four clique vertices will speed up the computation: Four cases should be run, one with each clique vertex receiving color 1. If all four cases are unsatisfiable, we've proved that $f(n) \geq r$ or $g(n) < r$, respectively. Binary search with different values of r will identify the optimum.

For speedier $g(n)$, first find a maximum independent set instead of a complete 4-coloring; then notice that the colorings for $f(n)$ already achieve this maximum.

The results turn out to be $f(n) = (2, 2, 3, 4, 5, 7, 7, 7, 8, 9, 10, 12, 12, 12)$ for $n = (3, 4, \ldots, 16)$, and $g(n) = (4, 6, 10, 13, 17, 23, 28, 35, 42, 50, 58, 68, 77, 88)$.

18. Assuming that $n \geq 4$, first assign to vertex (j, k) the following "default color": $1 + (j + k) \bmod 3$ if $j \leq k$; $1 + (j + k + 1 - n) \bmod 3$ if $k < j/2$; otherwise $1 + (j + k + 2 - n) \bmod 3$. Then make the following changes to exceptional vertices: Vertex $(1, 0)$ is colored 2 if $n \bmod 6 = 0$ or 5, otherwise 3. Vertex $(n, n-1)$ is colored 4. For $k \leftarrow 0$ up to $n-2$, change the color of vertex (n, k) to 4, if its default color matches vertex $(0, 0)$ when $k \leq n/2$ or vertex $(1, 0)$ when $k > n/2$. And make final touchups for $1 \leq j < n/2$, depending again on $n \bmod 6$:

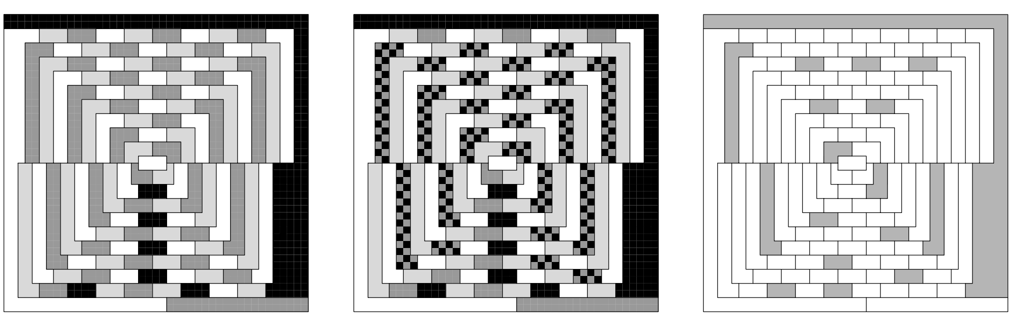
Case 0: Give color 4 to vertex $(2j, j-1)$ and color 1 to vertex $(2j+1, j)$.

Case 1: Give color 4 to vertex $(2j, j)$ and color 2 to vertex $(2j+1, j)$.

Case 2: Give color 4 to vertex $(2j, j)$ and color 1 to vertex $(2j+1, j)$. Also give $(n, n-2)$ the color 1 and $(n-1, n-3)$ the color 4.

Cases 3, 4, 5: Give color 4 to vertex $(2j+1, j)$.

For example, the coloring for the case $n = 10$ (found by Bryant) is shown in Fig. A–5(a).



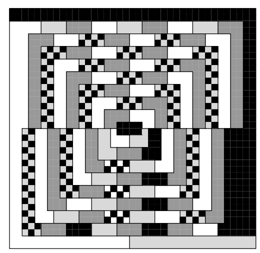
(a) (b) (c)

Fig. A–5. Colorings and kernels of McGregor's graph.

The color distribution is $(\lfloor n^2/3 \rfloor, \lfloor n^2/3 \rfloor, \lfloor n^2/3 \rfloor, 5k) + ((0, 1, k, -1),\ (1, k, 1, 0),\ (-1, k+1, 1, 2),\ (0, k, 1, 2),\ (1, k+1, 1, 2),\ (0, 2, k+1, 3))$, for $n \bmod 6 = (0, 1, 2, 3, 4, 5)$, $k = \lfloor n/6 \rfloor$. Since this construction achieves all of the optimum values for $f(n)$ and $g(n)$, when $n \leq 16$, it probably is optimum for all n. Moreover, the value of $g(n)$ agrees with the size of the maximum independent set in all known cases. A further conjecture is that the maximum independent set is *unique*, whenever $n \bmod 6 = 0$ and $n > 6$.

Laurier
mcgregor(n)
symmetric threshold functions
symmetry breaking
Binary search
maximum independent set
Bryant
maximum independent set

19. Use the clauses of *mcgregor* (n), together with $(v_1 \vee v_2 \vee v_3 \vee \bar{v}_x) \wedge (v_1 \vee v_2 \vee v_4 \vee \bar{v}_x) \wedge (v_1 \vee v_3 \vee v_4 \vee \bar{v}_x) \wedge (v_2 \vee v_3 \vee v_4 \vee \bar{v}_x)$ for each vertex, together with clauses from (20) and (21) that require at least r of the vertices v_x to be true. Also assign unique colors to the four clique vertices. (One assignment, not four, is sufficient to break symmetry here, because $h(n)$ is a more symmetrical property than $f(n)$ or $g(n)$.) These clauses are satisfiable if and only if $h(n) \geq r$. The SAT computation goes faster if we also provide clauses that require each color class to be a *kernel* (see exercise 21).



The values $h(n) = (1, 3, 4, 8, 9, 13)$ for $n = (3, 4, \ldots, 8)$ are readily obtained in this way. Furthermore, if we extend color class 4 in the construction of answer 18 to a suitable kernel, we find $h(9) \geq 17$ and $h(10) \geq 23$. The resulting diagram for $n = 10$, illustrated in Fig. A–5(b), nicely exhibits 2^{23} solutions to McGregor's original coloring problem, all at once.

A good SAT solver also shows that $h(9) \leq 18$ and $h(10) \leq 23$, thus proving that $h(10) = 23$. And Armin Biere's solver proved in 2013 that $h(9) = 18$, by discovering the surprising solution shown here. (This exercise was inspired by Frank Bernhart, who sent a diagram like Fig. A–5(b) to Martin Gardner in 1975; his diagram achieved 2^{21} solutions.)

20. Arrange the vertices (j, k) of answer 15 in the following order $v_0, v_1, \ldots$: $(n, n-1)$; $(0, n-1)$, $(0, n-2)$, $\ldots$, $(0, 0)$; $(1, n-1)$, $(1, n-2)$, $\ldots$, $(1, 1)$; $\ldots$; $(n-2, n-1)$, $(n-2, n-2)$; $(n-1, n-2)$, $(n-2, n-3)$, $\ldots$, $(2, 1)$; $(n-1, n-1)$; $(2, 0)$, $(3, 1)$, $\ldots$, $(n, n-2)$; $(3, 0)$, $(4, 1)$, $\ldots$, $(n, n-3)$; $(1, 0)$; $(4, 0)$, $\ldots$, $(n, n-4)$; $\ldots$; $(n-1, 0)$, $(n, 1)$; $(n, 0)$. Then if $V_t = \{v_0, \ldots, v_{t-1}\}$, let the "frontier" F_t consist of all vertices $\in V_t$ that have at least one neighbor $\notin V_t$. We can assume that (v_0, v_1, v_2) are colored $(0, 1, 2)$, because they are part of the 4-clique.

All 4-colorings of V_t that have a given sequence of colors on F_t can be enumerated if we know the corresponding counts for F_{t-1}. The stated ordering ensures that F_t never will contain more than $2n-1$ elements; in fact, at most 3^{2n-2} sequences of colors are feasible, for any given t. Since 3^{18} is less than 400 million, it's quite feasible to do these incremental calculations. The total (obtained with about 6 gigabytes of memory and after about 500 gigamems of computation) turns out to be 898,431,907,970,211.

This problem is too large to be handled efficiently by BDD methods when $n = 10$, but BDD calculations for $n \leq 8$ can be used to check the algorithm. The frontiers essentially represent level-by-level slices of a QDD for this problem. The 4-coloring counts for $3 \leq n \leq 9$ are respectively 6, 99, 1814, 107907, 9351764, 2035931737, 847019915170.

21. With one Boolean variable v for every vertex of a graph G, the kernels are characterized by the clauses (i) $\bar{u} \vee \bar{v}$ whenever $u \frac{\quad}{} v$; (ii) $v \vee \bigvee_{u \frac{\quad}{} v} u$ for all v. Adding to these the clauses for the symmetric threshold function $S_{\leq r}(x_1, \ldots, x_N)$, we can find the least r for which all clauses are satisfiable. The graph of Fig. 33 yields satisfiability for $r = 17$; and one of its 46 kernels of size 17 is shown in Fig. A–5(c).

[BDD methods are slower for this problem; but they enumerate *all* 520,428,275,749 of the kernels, as well as the generating function $46z^{17} + 47180z^{18} + \cdots + 317z^{34} + 2z^{35}$.]

22. Eight colors are needed. The coloring $\begin{smallmatrix}12771\\22788\\33668\\34655\\14451\end{smallmatrix}$ is "balanced," with each color used at least thrice.

23. Writing k for x_k and ${}^{k}_{j}$ for s^k_j, the clauses from (18)–(19) are ${}^{\bar1}_{1}{}^{1}_{2}$, ${}^{\bar1}_{2}{}^{1}_{3}$, ${}^{\bar2}_{1}{}^{2}_{2}$, ${}^{\bar2}_{2}{}^{2}_{3}$, ${}^{\bar3}_{1}{}^{3}_{2}$, ${}^{\bar3}_{2}{}^{3}_{3}$, ${}^{\bar4}_{1}{}^{4}_{2}$, ${}^{\bar4}_{2}{}^{4}_{3}$; $\bar1{}^{1}_{1}$, $\bar2{}^{1}_{2}$, $\bar3{}^{1}_{3}$, $\bar2{}^{\bar1}_{1}{}^{2}_{1}$, $\bar3{}^{\bar1}_{2}{}^{2}_{2}$, $\bar4{}^{\bar1}_{3}{}^{2}_{3}$, $\bar3{}^{\bar2}_{1}{}^{3}_{1}$, $\bar4{}^{\bar2}_{2}{}^{3}_{2}$, $\bar5{}^{\bar2}_{3}{}^{3}_{3}$, $\bar4{}^{\bar3}_{1}{}^{4}_{1}$, $\bar5{}^{\bar3}_{2}{}^{4}_{2}$, $\bar6{}^{\bar3}_{3}{}^{4}_{3}$, $\bar5{}^{\bar4}_{1}$, $\bar6{}^{\bar4}_{2}$, $\bar7{}^{\bar4}_{3}$.

Similarly, (20) and (21) define the clauses $\bar7{}^{6}_{1}$, $\bar6{}^{6}_{1}$, $\bar6\bar7{}^{6}_{2}$; $\bar5{}^{5}_{1}$, $\bar4{}^{5}_{1}$, $\bar4\bar5{}^{5}_{2}$; $\bar3{}^{4}_{1}$, $\bar2{}^{4}_{1}$, $\bar2\bar3{}^{4}_{2}$; $\bar1{}^{3}_{1}$, ${}^{\bar6}_{1}{}^{3}_{1}$, ${}^{\bar6}_{1}\bar1{}^{3}_{2}$, ${}^{\bar6}_{2}{}^{3}_{2}$, ${}^{\bar6}_{2}\bar1{}^{3}_{3}$; ${}^{\bar5}_{1}{}^{2}_{1}$, ${}^{\bar4}_{1}{}^{2}_{1}$, ${}^{\bar5}_{2}{}^{2}_{2}$, ${}^{\bar4}_{2}{}^{2}_{2}$, ${}^{\bar4}_{1}{}^{\bar5}_{1}{}^{2}_{2}$, ${}^{\bar4}_{1}{}^{\bar5}_{2}{}^{2}_{3}$, ${}^{\bar4}_{2}{}^{\bar5}_{1}{}^{2}_{3}$, ${}^{\bar4}_{2}{}^{\bar5}_{2}{}^{2}_{4}$; ${}^{\bar2}_{4}{}^{\bar3}_{1}$, ${}^{\bar2}_{3}{}^{\bar3}_{2}$, ${}^{\bar2}_{2}{}^{\bar3}_{3}$. So

symmetry breaking
kernel
McGregor
Biere
Bernhart
Gardner
frontier
gigamems
BDD
QDD
symmetric threshold function
BDD
generating function

this tree-based method apparently needs one more variable and two more clauses when $(n,r)=(7,4)$. But the next exercise shows that (18) and (19) don't really win!

24. (a) The clause $(\bar b_1^2 \vee \bar b_r^3)$ appears only if $t_3=r$; and $t_3 \le n/2$.

(b) For example, $t_3=\min(r,4)<r$ when $n=11$ and $r=5$.

(c) In this case t_k is the number of leaves below node k, and the only auxiliary variables that survive pure literal elimination are $b_{t_k}^k$. We're left with just $n-1$ surviving clauses, namely $(\bar b_{t_{2k}}^{2k} \vee \bar b_{t_{2k+1}}^{2k+1} \vee b_{t_k}^k)$ for $1<k<n$, plus $(\bar b_{t_2}^2 \vee \bar b_{t_3}^3)$.

(d) If $2^k \le n \le 2^k+2^{k-1}$ we have $(n',n'')=(n-2^{k-1},2^{k-1})$; on the other hand if $2^k+2^{k-1} \le n \le 2^{k+1}$ we have $(n',n'')=(2^k,n-2^k)$. (Notice that $n'' \le n' \le 2n''$.)

(e) No pure literals are removed in this completely balanced case (which is the easiest to analyze). We find $a(2^k,2^{k-1})=(k-1)2^k$ and $c(2^k,2^{k-1})=(2^{k-2}+k-1)2^k$.

(f) One can show that $a(n,r)=\bigl(r\le n''?\ b(n',r)+b(n'',r)\colon r\le n'?\ b(n',n'')+b(n'',n'')\colon b(n',n-r)+b(n'',n-r)\bigr)$, where $b(1,1)=0$ and $b(n,r)=r+b(n',\min(r,n'))+b(n'',\min(r,n''))$ for $n\ge 2$. Similarly, $c(n,r)=\bigl(r\le n''?\ r+f(n',0,r)+f(n'',0,r)\colon r\le n'?\ n''+f(n',r-n'',r)+f(n'',0,n'')\colon n-r+f(n',r-n'',n')+f(n'',r-n',n'')\bigr)$, where $f(n,l,r)=\sum_{k=l+1}^{r}\min(k+1,n''+1,n+1-k)+\bigl(r\le n''?\ r+f(n',0,r)+f(n'',0,r)\colon r\le n'?\ n''+f(n',0,r)+f(n'',0,n'')\colon r<n?\ n-r+f(n',0,n')+f(n'',0,n'')\colon r-l<n''?\ f(n',n'-r+l,n')+f(n'',n''-r+l,n'')\colon r-l<n'?\ f(n',n'-r+l,n')+f(n'',0,n'')\colon f(n',0,n')+f(n'',0,n'')\bigr)$ for $n\ge 2$ and $f(1,0,1)=0$. The desired results follow by induction from these recurrence relations.

Incidentally, ternary branching can give further savings. We can, for example, handle the case $n=6$, $r=3$ with 17 clauses in the 6 variables b_1^2, b_2^2, b_3^2, b_1^3, b_2^3, b_3^3.

25. From (18) and (19) we obtain $5n-12$ clauses in $2n-4$ variables, with a simple lattice-like structure. But (20) and (21) produce a more complex tree-like pattern, with $2n-4$ variables and with $\lfloor n/2\rfloor$ nodes covering just two leaves. So we get $\lfloor n/2\rfloor$ nodes with 3 clauses, $n \bmod 2$ nodes with 5 clauses, $\lceil n/2\rceil$ nodes with 7 clauses, and 2 clauses from (21), totalling $5n-12$ as before (assuming that $n>3$). In fact, all but $n-2$ of the clauses are binary in both cases.

26. Imagine the boundary conditions $s_j^0=1$, $s_j^{r+1}=0$, $s_0^k=0$, for $1\le j\le n-r$ and $1\le k\le r$. The clauses say that $s_1^k\le\cdots\le s_{n-r}^k$ and that $x_{j+k}s_j^k\le s_j^{k+1}$; so the hint follows by induction on j and k.

Setting $j=n-r$ and $k=r+1$ shows that we cannot satisfy the new clauses when $x_1+\cdots+x_n\ge r+1$. Conversely, if we can satisfy F with $x_1+\cdots+x_n\le r$ then we can satisfy (18) and (19) by setting $s_j^k\leftarrow[x_1+\cdots+x_{j+k-1}\ge k]$.

27. Argue as in the previous answer, but imagine that $b_0^k=1$, $b_{r+1}^1=0$; prove the hint by induction on j and $n-k$ (beginning with $k=n-1$, then $k=n-2$, and so on).

28. For example, the clauses for $\bar x_1+\cdots+\bar x_n\le n-1$ when $n=5$ are $(x_1\vee s_1^1)$, $(x_2\vee\bar s_1^1\vee s_1^2)$, $(x_3\vee\bar s_1^2\vee s_1^3)$, $(x_4\vee\bar s_1^3\vee s_1^4)$, $(x_5\vee\bar s_1^4)$. We may assume that $n\ge 4$; then the first two clauses can be replaced by $(x_1\vee x_2\vee s_1^2)$, and the last two by $(x_{n-1}\vee x_n\vee\bar s_1^{n-2})$, yielding $n-2$ clauses of length 3 in $n-3$ auxiliary variables.

29. We can assume that $1\le r_1\le\cdots\le r_n=r<n$. Sinz's clauses (18) and (19) actually do the job nicely if we also assert that s_j^k is false whenever $k=r_i+1$ and $j=i-r_i$.

30. The clauses now are $(\bar s_j^k\vee s_{j+1}^k)$, $(\bar x_{j+k}\vee\bar s_j^k\vee s_j^{k+1})$, $(s_j^k\vee\bar s_j^{k+1})$, $(x_{j+k}\vee s_j^k\vee\bar s_{j+1}^k)$, hence they define the quantities $s_j^k=[x_1+\cdots+x_{j+k-1}\ge k]$; implicitly $s_0^k=s_j^{r+1}=0$ and $s_j^0=s_{n-r+1}^k=1$. The new clauses in answer 23 are ${}^{1\,\bar2}_{1\,1}$, ${}^{2\,\bar3}_{1\,1}$, ${}^{3\,\bar4}_{1\,1}$, ${}^{1\,\bar2}_{2\,2}$, ${}^{2\,\bar3}_{2\,2}$, ${}^{3\,\bar4}_{2\,2}$, ${}^{1\,\bar2}_{3\,3}$, ${}^{2\,\bar3}_{3\,3}$, ${}^{3\,\bar4}_{3\,3}$; $1^{\bar1}_{1}$, $2^{\bar2}_{1}$, $3^{\bar3}_{1}$, $4^{\bar4}_{1}$, $2^{1\,\bar1}_{1\,2}$, $3^{2\,\bar2}_{1\,2}$, $4^{3\,\bar3}_{1\,2}$, $5^{4\,\bar4}_{1\,2}$, $3^{1\,\bar1}_{2\,3}$, $4^{2\,\bar2}_{2\,3}$, $5^{3\,\bar3}_{2\,3}$, $6^{4\,\bar4}_{2\,3}$, 4^{1}_{3}, 5^{2}_{3}, 6^{3}_{3}, 7^{4}_{3}.

binary recurrence relations
recurrence relations
Sinz

With (20) and (21) we can identify b'^{k}_{j} with $\bar b^{k}_{l_k+1-j}$, when $l_k > 1$ leaves are below node k. Then b^k_j is true if *and only if* the leaves below k have j or more 1s. For example, answer 23 gets the new clauses $7^{\bar6}_{2}$, $6^{\bar6}_{2}$, $67^{\bar6}_{1}$; $5^{\bar5}_{2}$, $4^{\bar5}_{2}$, $45^{\bar5}_{1}$; $3^{\bar4}_{2}$, $2^{\bar4}_{2}$, $23^{\bar4}_{1}$; $1^{\bar3}_{3}$, ${}^{6}_{2}{}^{\bar3}_{3}$, $1^{6}_{2}{}^{\bar3}_{2}$, ${}^{6}_{1}{}^{\bar3}_{2}$, $1^{6}_{1}{}^{\bar3}_{1}$; ${}^{4}_{2}{}^{\bar2}_{4}$, ${}^{5}_{2}{}^{\bar2}_{4}$, ${}^{4}_{1}{}^{\bar2}_{3}$, ${}^{4}_{2}{}^{5}_{2}{}^{\bar2}_{3}$, ${}^{5}_{1}{}^{\bar2}_{3}$, ${}^{4}_{1}{}^{5}_{2}{}^{\bar2}_{2}$, ${}^{4}_{2}{}^{5}_{1}{}^{\bar2}_{2}$, ${}^{4}_{1}{}^{5}_{1}$; ${}^{2}_{4}{}^{3}_{1}$, ${}^{2}_{3}{}^{3}_{2}$, ${}^{2}_{2}{}^{3}_{3}$.

Furthermore, (20) and (21) can be unified in the same way with the weaker constraints $r' \le x_1+\cdots+x_n \le r$. If we want, say, $2 \le x_1+\cdots+x_7 \le 4$, we can simply replace the final four clauses of the previous paragraph by ${}^{4}_{1}{}^{5}_{1}{}^{\bar2}_{1}$, ${}^{2}_{2}{}^{3}_{1}$, ${}^{2}_{1}{}^{3}_{2}$. Under the conventions of (18) and (19), by contrast, these weaker constraints would generate a comparable number of new clauses, namely ${}^{1}_{1}{}^{\bar2}_{1}$, ${}^{1}_{2}{}^{\bar2}_{2}$, ${}^{1}_{3}{}^{\bar2}_{3}$, ${}^{1}_{4}{}^{\bar2}_{4}$, ${}^{1}_{5}{}^{\bar2}_{5}$ and $1^{\bar1}_{1}$, $2^{\bar2}_{1}$, $3^{2}_{1}{}^{\bar2}_{2}$, $3^{1}_{2}{}^{\bar1}_{3}$, $4^{2}_{2}{}^{\bar2}_{3}$, $4^{1}_{3}{}^{\bar1}_{4}$, $5^{2}_{3}{}^{\bar2}_{4}$, $5^{1}_{4}{}^{\bar1}_{5}$, $6^{2}_{4}{}^{\bar2}_{5}$, 6^{1}_{5}, 7^{2}_{5}; but those clauses involve the new variables ${}^{1}_{4}$, ${}^{1}_{5}$, ${}^{2}_{4}$, ${}^{2}_{5}$.

31. We can use the constraints on the second line of (10), together with the constraints of exercise 30 that force $x_1+\cdots+x_n = r$. Then we seek n for which this problem is satisfiable, while the same problem with $x_n = 0$ is not. The following small values can be used to check the calculations:

$r =$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
$F_3(r) =$	1	2	4	5	9	11	13	14	20	24	26	30	32	36	40	41	51	54	58	63	71	74	82	84	92	95	100
$F_4(r) =$	1	2	3	5	6	8	9	10	13	15	17	19	21	23	25	27	28	30	33	34	37	40	43	45	48	50	53
$F_5(r) =$	1	2	3	4	6	7	8	9	11	12	13	14	16	17	18	19	24	25	27	28	29	31	33	34	36	37	38
$F_6(r) =$	1	2	3	4	5	7	8	9	10	12	13	14	15	17	18	19	20	22	23	24	25	26	29	32	33	35	36

Furthermore, significant speedup is possible if we also make use of previously computed values $F_t(1), \ldots, F_t(r-1)$. For example, when $t = 3$ and $r \ge 5$ we must have $x_{a+1}+\cdots+x_{a+8} \le 4$ for $0 \le a \le n-8$, because $F_3(5) = 9$. These additional subinterval constraints blend beautifully with those of exercise 30, because $x_{a+1}+\cdots+x_{a+p} \le q$ for $0 \le a \le n-p$ implies $\bar s^{k}_{b+p-q} \vee s^{k-q}_{b}$ for $0 \le b \le n+1-p+q-r$ and $q < k \le r$.

We can also take advantage of left-right symmetry by appending the unit clause $\bar s^{\lceil r/2\rceil}_{\lceil (n-r)/2\rceil}$ when r is odd; $s^{r/2}_{n/2-r/2+1}$ when n and r are both even.

Suitable benchmark examples arise when computing, say, $F_3(27)$ or $F_4(36)$. But for large cases, general SAT-based methods do not seem to compete with the best special-purpose backtrack routines. For example, Gavin Theobald and Rodolfo Niborski have obtained the value $F_3(41) = 194$, which seems well beyond the reach of these ideas.

[See P. Erdös and P. Turán, *J. London Math. Soc.* (2) **11** (1936), 261–264; errata, **34** (1959), 480; S. S. Wagstaff, Jr., *Math. Comp.* **26** (1972), 767–771.]

32. Use (15) and (16), and optionally (17), but omit variable v_j unless $j \in L(v)$.

33. To double-color a graph with k colors, change (15) to the set of k clauses $v_1 \vee \cdots \vee v_{j-1} \vee v_{j+1} \vee v_k$, for $1 \le j \le k$; similarly, $\binom{k}{2}$ clauses of length $k-2$ will yield a triple coloring. Small examples reveal that C_{2l+1} for $l \ge 2$ can be double-colored with five colors: $\{1,2\}\bigl(\{3,4\}\{5,1\}\bigr)^{l-1}\{2,3\}\{4,5\}$; furthermore, seven colors suffice for triple coloring when $l \ge 3$: $\{1,2,3\}\bigl(\{4,5,6\}\{7,1,2\}\bigr)^{l-2}\{3,4,5\}\{6,7,1\}\{2,3,4\}\{5,6,7\}$. The following exercise proves that those colorings are in fact optimum.

34. (a) We can obviously find a q-tuple coloring with $q\chi(G)$ colors. And McGregor's graph has a four-clique, hence $\chi^*(G) \ge 4$.

(b) Any q-tuple coloring with p colors yields a solution to the fractional exact cover problem, if we let $\lambda_j = \sum_{i=1}^{p}[S_j$ is the set of vertices colored $i]/q$. Conversely, the theory of linear equalities tells us that there is always an optimum solution with rational $\{\lambda_1, \ldots, \lambda_N\}$; such a solution yields a q-tuple coloring when each $q\lambda_j$ is an integer.

unary encoding
cardinality constraints, intervals
subinterval constraints
cardinality constraints, subintervals
symmetry
benchmark
backtrack
Theobald
Niborski
Erdös
Turán
Wagstaff
clique

(c) $\chi^*(C_n) = \chi(C_n) = 2$ when n is even; and $\chi^*(C_{2l+1}) \le 2 + 1/l = n/\alpha(C_{2l+1})$, because there's an l-tuple coloring with n colors as in the previous exercise. Also $\chi^*(G) \ge n/\alpha(G)$ in general: $n = \sum_v \sum_j \lambda_j [v \in S_j] = \sum_j \lambda_j |S_j| \le \alpha(G) \sum_j \lambda_j$.

(d) For the hint, let $S = \{v_1, \ldots, v_l\}$ where vertices are sorted by their colors. Since vertex v_j belongs to C_i with $|C_i| \ge |\{v_j, \ldots, v_l\}|$, we have $t_{v_j} \le 1/(l+1-j)$.

So $\chi(G) \le k = \sum_v t_v = \sum_v t_v \sum_j \lambda_j [v \in S_j] = \sum_j \lambda_j \sum_v t_v [v \in S_j] \le \sum_j \lambda_j H_{\alpha(G)}$.

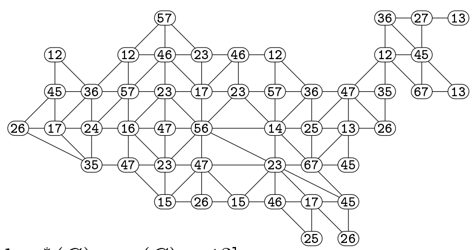
[See David S. Johnson, *J. Computer and System Sci.* **9** (1974), 264–269; L. Lovász, *Discrete Math.* **13** (1975), 383–390. The concept of fractional covering is due to A. J. W. Hilton, R. Rado, and S. H. Scott, *Bull. London Math. Soc.* **5** (1973), 302–306.]

Johnson
Lovász
Hilton
Rado
Scott
wheel
degree
WalkSAT
symmetry

35. (a) The double coloring below proves that $\chi^*(G) \ge 7/2$; and it is optimum because NV and its neighbors induce the wheel W_6. (Notice that $\chi^*(W_n) = 1 + \chi^*(C_{n-1})$.)

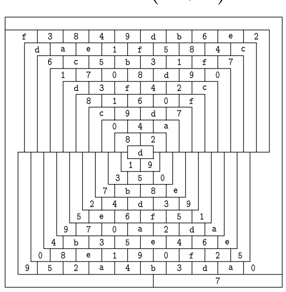
(b) By part (c) of the previous exercise, $\chi^*(G) \ge 25/4$. Furthermore there is a quadruple coloring with 25 colors:

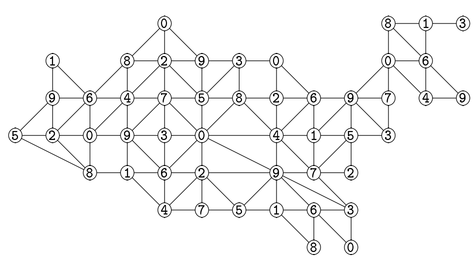
```
AEUY ABUV BCVW CDWX DEXY
AEFJ ABFG BCGH CDHI DEIJ
FJKO FGKL GHLM HIMN IJNO
KOPT KLPQ LMQR MNRS NOST
PTUY PQUV QRVW RSWX STXY
```



[Is $\overline{C_5 \boxtimes C_5}$ the smallest graph for which $\chi^*(G) < \chi(G) - 1$?]

36. A few more binary color constraints analogous to (16) yield the corresponding SAT problem. We can also assume that the upper right corner is colored 0, because that region touches $n + 4 = 14$ others; at least $n + 6$ colors are needed. The constraints elsewhere aren't very tight (see exercise 38(b)); thus we readily obtain an optimum radio coloring with $n + 6$ colors for the McGregor graphs of all orders $n > 4$, such as the one below. An $(n+7)$th color is necessary and sufficient when $n = 3$ or 4.





37. The 10-coloring shown here is optimum, because Missouri (MO) has degree 8.

38. By looking at solutions for $n = 10$, say, which can be obtained quickly via Algorithm W (WalkSAT), it's easy to discover patterns that work in general: (a) Let (x, y) have color $(2x + 4y) \bmod 7$. (Seven colors are clearly necessary when $n \ge 3$.) (b) Let (x, y, z) have color $(2x + 6y) \bmod 9$. (Nine colors are clearly necessary when $n \ge 4$.)

39. Let $f(n)$ denote the fewest consecutive colors. SAT solvers readily verify that $f(n) = (1, 3, 5, 7, 8, 9)$ for $n = (0, 1, 2, 3, 4, 5)$. Furthermore we can exploit symmetry to show that $f(6) > 10$: One can assume that 000000 is colored 0, and that the colors of 000001, ..., 100000 are increasing; that leaves only three possibilities for each of the

latter. Finally, we can verify that $f(6) = 11$ by finding a solution that uses only the colors $\{0, 1, 3, 4, 6, 7, 9, 10\}$.

But $f(7)$ is known only to be ≥ 11 and ≤ 15.

[$L(2,1)$ labelings were named by J. R. Griggs and R. K. Yeh, who initiated the theory in *SIAM J. Discrete Math.* **5** (1992), 586–595. The best known upper bounds, including the fact that $f(2^k - k - 1) \leq 2^k$, were obtained by M. A. Whittlesey, J. P. Georges, and D. W. Mauro, who also solved exercise 38(a); see *SIAM J. Discrete Math.* **8** (1995), 499–506.]

40. No; the satisfiable cases are $z = 0$, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 15, 21. [The statement would have been true if we'd also required $(x_m \vee \cdots \vee x_2) \wedge (y_n \vee \cdots \vee y_2)$.]

41. First there are mn ANDs to form $x_i y_j$. A bin that contains t bits initially will generate $\lfloor t/2 \rfloor$ carries for the next bin, using $(t-1)/2$ adders. (For example, $t = 6$ will invoke 2 full adders and one half adder.) The respective values of t for $bin[2]$, $bin[3]$, ..., $bin[m+n+1]$ are (1, 2, 4, 6, ..., $2m-2$, $2m-1$, ..., $2m-1$, $2m-2$, $2m-3$, ..., 5, 3, 1), with $n-m$ occurrences of $2m-1$. That makes a total $mn - m - n$ full adders and m half adders; altogether we get $mn + 2(mn - m - n) + m$ instances of AND, $mn - m - n$ instances of OR, and $2(mn - m - n) + m$ instances of XOR.

42. Ternary XOR requires quaternary clauses, but ternary clauses suffice for median:

$(t \vee u \vee v \vee \bar{x})$	$(t \vee u \vee \bar{v} \vee x)$	$(t \vee u \vee \bar{y})$	$(\bar{t} \vee \bar{u} \vee y)$
$(t \vee \bar{u} \vee \bar{v} \vee \bar{x})$	$(t \vee \bar{u} \vee v \vee x)$	$(t \vee v \vee \bar{y})$	$(\bar{t} \vee \bar{v} \vee y)$
$(\bar{t} \vee u \vee \bar{v} \vee \bar{x})$	$(\bar{t} \vee u \vee v \vee x)$	$(u \vee v \vee \bar{y})$	$(\bar{u} \vee \bar{v} \vee y)$
$(\bar{t} \vee \bar{u} \vee v \vee \bar{x})$	$(\bar{t} \vee \bar{u} \vee \bar{v} \vee x)$		

These clauses specify respectively that $x \leq t \oplus u \oplus v$, $x \geq t \oplus u \oplus v$, $y \leq \langle tuv \rangle$, $y \geq \langle tuv \rangle$.

43. $x = y = 3$ works when $n = 2$, but the cases $3 \leq n \leq 7$ are unsatisfiable. We can use $x = 3(2^{n-2} + 1)$, $y = 7(2^{n-3} + 1)$ for all $n \geq 8$. (Such solutions seem to be quite rare. Another is $x = 3227518467$, $y = 3758194695$ when $n = 32$.)

44. First scout the territory quickly by looking at all $\binom{N+1}{2} \approx 660$ billion cases with at most six zeros in x or y; here $N = \binom{32}{26} + \binom{32}{27} + \cdots + \binom{32}{32}$. This uncovers the remarkable pair $x = 2^{32} - 2^{26} - 2^{22} - 2^{11} - 2^8 - 2^4 - 1$, $y = 2^{32} - 2^{11} + 2^8 - 2^4 + 1$, whose product is $2^{64} - 2^{58} - 2^{54} - 2^{44} - 2^{33} - 2^8 - 1$. Now a SAT solver finishes the job by showing that the clauses for 32×32 bit multiplication are unsatisfiable in the presence of the further constraint $\bar{x}_1 + \cdots + \bar{x}_{32} + \bar{y}_1 + \cdots + \bar{y}_{32} + \bar{z}_1 + \cdots + \bar{z}_{64} \leq 15$. (The LIFO version of the clauses worked much faster than FIFO in the author's experiments with Algorithm L. Symmetry was broken by separate runs with $x_k \ldots x_1 = 01^{k-1}$, $y_k \ldots y_1 = 1^k$.)

45. Use the clauses for $xy = z$ in the factorization problem, with $m = \lfloor t/2 \rfloor$, $n = \lceil t/2 \rceil$, and $x_j = y_j$ for $1 \leq j \leq m$; append the unit clause $(\bar{y}_n)$ if $m < n$.

46. The two largest, 285000288617375^2 and 301429589329949^2, have 97 bits; the next square binary palindrome, 1178448744881657^2, has 101. [This problem is *not* easy for SAT solvers; number theory does much better. Indeed, there's a nice way to find all n-bit examples by considering approximately $2^{n/3}$ cases, because the rightmost $n/3$ bits of an $n/2$-bit number x force the other $n/6$ bits, if x^2 is palindromic. The first eight square binary palindromes were found by G. J. Simmons, *J. Recreational Math.* **5** (1972), 11–19; all 31 solutions up to 2^{95} were found by J. Schoenfield in 2009.]

47. Each wire has a "top" and a "bottom." There are $n + g + 2h$ tops of wires, and $m + 2g + h$ bottoms of wires. Hence the total number of wires is $n + g + 2h = m + 2g + h$, and we must have $n + h = m + g$.

Griggs
Yeh
Whittlesey
Georges
Mauro
carries
full adders
half adder
Tseytin encoding, half
Knuth
Symmetry was broken
factorization
unit clause
SAT solvers
number theory
Simmons
Schoenfield

48. The wires compute $q^1 \leftarrow q$, $q^2 \leftarrow q$, $x \leftarrow p \oplus q^1$, $y \leftarrow q^2 \oplus r$, $z \leftarrow x \oplus y$. Let p denote "p stuck at 1" while $\bar{p}$ denotes "p stuck at 0". The pattern $pqr = 000$ detects p, q^1, q^2, r, x, y, z; 001 detects p, q^1, q^2, $\bar{r}$, x, $\bar{y}$, $\bar{z}$; 010 detects p, $\bar{q}^1$, $\bar{q}^2$, r, $\bar{x}$, $\bar{y}$, z; 011 detects p, $\bar{q}^1$, $\bar{q}^2$, $\bar{r}$, $\bar{x}$, y, $\bar{z}$; 100 detects $\bar{p}$, q^1, q^2, r, $\bar{x}$, y, $\bar{z}$; 101 detects $\bar{p}$, q^1, q^2, $\bar{r}$, $\bar{x}$, $\bar{y}$, z; 110 detects $\bar{p}$, $\bar{q}^1$, $\bar{q}^2$, r, x, $\bar{y}$, $\bar{z}$; 111 detects $\bar{p}$, $\bar{q}^1$, $\bar{q}^2$, $\bar{r}$, x, y, z. Notice that the stuck-at faults for q aren't detectable (because $z = (p \oplus q) \oplus (q \oplus r) = p \oplus r$); but we can detect faults on its clones q^1, q^2. (In Fig. 34 the opposite happens.)

Three patterns such as $\{100, 010, 001\}$ suffice for all of the detectable faults.

49. One finds, for example, that the faults b_3^2, $\bar{c}_1^2$, $\bar{s}^2$, and $\bar{q}$ are detected *only* by the pattern $y_3y_2y_1x_2x_1 = 01111$; $\bar{a}_2^2$, $\bar{a}_3^2$, $\bar{b}_3^2$, $\bar{p}$, $\bar{c}_2^2$, $\bar{z}_5$ are detected only by 11011 or 11111.

All covering sets can be found by setting up a CNF with 99 positive clauses, one for each detectable fault; for example, the clause for $\bar{z}_5$ is $x_{27} \vee x_{31}$, while the clause for x_2^2 is $x_4 \vee x_5 \vee x_{12} \vee x_{13} \vee x_{20} \vee x_{21} \vee x_{28} \vee x_{29}$. We can find minimum covers from a BDD for these clauses, or by using a SAT solver with additional clauses such as (20) and (21) to limit the number of positive literals. Exactly fourteen sets of five patterns suffice, the most memorable being $\{01111, 10111, 11011, 11101, 11110\}$. (Indeed, every minimum set includes at least three of these five patterns.)

50. Primed variables for tarnished wires are x_2', b_2', b_3', s', p', q', z_3', c_2', z_4', z_5'. Those wires also have sharped variables $x_2^\sharp$, $b_2^\sharp$, $\ldots$, $z_5^\sharp$; and we need sharped variables $x_2^{1\sharp}$, $x_2^{3\sharp}$, $x_2^{4\sharp}$, $b_2^{1\sharp}$, $b_2^{2\sharp}$, $b_3^{1\sharp}$, $b_3^{2\sharp}$, $s^{1\sharp}$, $s^{2\sharp}$, $c_2^{1\sharp}$, $c_2^{2\sharp}$ for fanout wires. The primed variables are defined by clauses such as $(\bar{p}' \vee a_3) \wedge (\bar{p}' \vee b_2') \wedge (p' \vee \bar{a}_3 \vee \bar{b}_2')$, which corresponds to $p' \leftarrow a_3 \wedge b_2'$. Those clauses are appended to the 49 clauses listed after (23) in the text. Then there are two clauses (25) for nine of the ten primed-and-sharped variables; however, in the case of x_2 we use the unit clauses $(x_2') \wedge (\bar{x}_2)$ instead, because the variable $x_2^\sharp$ doesn't exist. There are five fanout clauses (26), namely $(\bar{x}_2^{1\sharp} \vee x_2^{3\sharp} \vee x_2^{4\sharp}) \wedge (\bar{b}_2^\sharp \vee b_2^{1\sharp} \vee b_2^{2\sharp}) \wedge \cdots \wedge (\bar{c}_2^\sharp \vee c_2^{1\sharp} \vee c_2^{2\sharp})$. There are eleven clauses $(\bar{x}_2^{3\sharp} \vee b_2^\sharp) \wedge (\bar{x}_2^{4\sharp} \vee b_3^\sharp) \wedge (\bar{b}_2^{1\sharp} \vee s^\sharp) \wedge \cdots \wedge (\bar{b}_3^{2\sharp} \vee z_5^\sharp) \wedge (\bar{c}_2^{2\sharp} \vee z_5^\sharp)$ for tarnished inputs to gates. And finally there's $(x_2^{1\sharp}) \wedge (z_3^\sharp \vee z_4^\sharp \vee z_5)$.

51. (The complete set of 196 patterns found by the author in 2013 included the inputs $(x, y) = (2^{32} - 1, 2^{31} + 1)$ and $(\lceil 2^{63/2} \rceil, \lceil 2^{63/2} \rceil)$ as well as the two number-theoretic patterns mentioned in the text. Long runs of carries are needed in the products.)

52. $(z_{1,2} \vee z_{2,2} \vee \cdots \vee z_{M,2}) \wedge (\bar{z}_{i,2} \vee \bar{q}_{i,1}) \wedge (\bar{z}_{i,2} \vee \bar{p}_{i,2}) \wedge (\bar{z}_{i,2} \vee \bar{q}_{i,3}) \wedge (\bar{z}_{i,2} \vee \bar{p}_{i,4}) \wedge \cdots \wedge (\bar{z}_{i,2} \vee \bar{q}_{i,20})$, for $1 \le i \le M$. The second subscript of z is k in the kth case, $1 \le k \le P$.

53. On the left is the binary expansion of π, and on the right is the binary expansion of e, 20 bits at a time (see Appendix A).

One way to define $f(x)$ for all 20-bit x is to write $\pi/4 = \sum_{k=1}^{\infty} u_k/2^{20k}$ and $e/4 = \sum_{l=1}^{\infty} v_l/2^{20l}$, where each u_k and v_l is a 20-bit number. Let k and l be smallest such that $x = u_k$ and $x = v_l$. Then $f(x) = [k \le l]$.

Equation (27) has actually been contrived to *sustain* an illusion of magic: Many simple Boolean functions are consistent with the data in Table 2, even if we require four-term DNFs of three literals each. But only two of them, like (27), have the additional property that they actually agree with the definition of $f(x)$ in the previous paragraph for ten more cases, using u_k up to $k = 22$ and v_l up to $l = 20$! One might almost begin to suspect that a SAT solver has discovered a deep new connection between π and e.

54. (a) The function $\bar{x}_1x_9x_{11}\bar{x}_{18} \vee \bar{x}_6\bar{x}_{10}\bar{x}_{12} \vee \bar{x}_4x_{10}\bar{x}_{12}$ matches all 16 rows of Table 2; but adding the 17th row makes a 3-term DNF impossible.

(b) 21 rows are impossible, but (27) satisfies 20 rows.

CNF
minimum covers
BDD
cardinality constraints
tarnished wires
Knuth
carries
pi, as source
e, as source
magic

(c) $\bar{x}_1\bar{x}_5\bar{x}_{12}x_{17} \vee \bar{x}_4x_8\bar{x}_{13}\bar{x}_{15} \vee \bar{x}_6\bar{x}_9\bar{x}_{12}x_{16} \vee \bar{x}_6\bar{x}_{13}\bar{x}_{16}x_{20} \vee x_{13}x_{14}\bar{x}_{16}$ does 28, which is max. (Incidentally, this problem makes no sense for sufficiently large M, because the equation $f(x) = 1$ probably does not have exactly 2^{19} solutions.)

55. Using (28)–(31) with $p_{i,j} = 0$ for all i and j, and also introducing clauses like (20) and (21) to ensure that $q_{i,1} + \cdots + q_{i,20} \le 3$, leads to solutions such as

$$f(x_1, \ldots, x_{20}) = \bar{x}_1\bar{x}_7\bar{x}_8 \vee \bar{x}_2\bar{x}_3\bar{x}_4 \vee \bar{x}_4\bar{x}_{13}\bar{x}_{14} \vee \bar{x}_6\bar{x}_{10}\bar{x}_{12}.$$

(There are no monotone *increasing* solutions with ≤ 4 terms of *any* length.)

56. We can define f consistently from only a subset of the variables if and only if no entry on the left agrees with any entry on the right, when restricted to those coordinate positions. For example, the first 10 coordinates do not suffice, because the top entry on the left begins with the same 10 bits as the 14th entry on the right. The first 11 coordinates do suffice (although two entries on the right actually agree in their first 12 bits).

Let the vectors on the left be u_k and v_l as in answer 53, and form the 256×20 matrix whose rows are $u_k \oplus v_l$ for $1 \le k, l \le 16$. We can solve the stated problem if and only if we can find five columns for which that matrix isn't 00000 in any row. This is the classical *covering problem* (but with rows and columns interchanged): We want to find five columns that cover every row.

In general, such an $m \times n$ covering problem corresponds to an instance of SAT with m clauses and n variables x_j, where x_j means "select column j." The clause for a particular row is the OR of the x_j for each column j in which that row contains 1. For example, in Table 2 we have $u_1 \oplus v_1 = 01100100111101111000$, so the first clause is $x_2 \vee x_3 \vee x_6 \vee \cdots \vee x_{17}$. To cover with at most five columns, we add suitable clauses according to (20) and (21); this gives 396 clauses of total length 2894, in 75 variables.

(Of course $\binom{20}{5}$ is only 15504; we don't need a SAT solver for this simple task! Yet Algorithm D needs only 578 kilomems, and Algorithm C finds an answer in 353 Kμ.)

There are 12 solutions: We can restrict to coordinates x_j for j in $\{1,4,15,17,20\}$, $\{1,10,15,17,20\}$, $\{1,15,17,18,20\}$, $\{4,6,7,10,12\}$, $\{4,6,9,10,12\}$, $\{4,6,10,12,19\}$, $\{4,10,12,15,19\}$, $\{5,7,11,12,15\}$, $\{6,7,8,10,12\}$, $\{6,8,9,10,12\}$, $\{7,10,12,15,20\}$, or $\{8,15,17,18,20\}$. (Incidentally, BDD methods show that the number of solutions to the covering problem has the generating function $12z^5 + 994z^6 + 13503z^7 + \cdots + 20z^{19} + z^{20}$, counting by the size of the covering set.)

57. Table 2 specifies a partially defined function of 20 Boolean variables, having $2^{20} - 32$ "don't-cares." Exercise 56 shows how to embed it in a partially defined function of only 5 Boolean variables, in twelve different ways. So we have twelve different truth tables:

11110110	0*1*010*	10000111	10*0*1*0	00100101	11110*0*	1011****	**0**00*
011*011*	1*110100	10*001*1	1000**10	100*1**0	11*00010	1100**0*	*0**0101
011*1*11	010*100*	10*0*000	*101*011	**1*1000	1*101100	1*100*10	0*****1*
10101110	0*100*1*	1*001*00	1**00***	1*1*1*10	10001100	0*101*1*	**1*0*10
10101110	0*1*0*10	1*0*1*00	0**01***	1*01*00*	1101*0*0	0011*11*	1*100*0*
1*01110*	00**110*	11**0*00	10*****0	001*1001	*1**1*1*	11*0*010	01011001

And the tenth of these yields $f(x) = ((x_8 \oplus (x_9 \vee x_{10})) \vee ((x_6 \vee x_{12}) \oplus \bar{x}_{10})) \oplus x_{12}$.

58. These clauses are satisfiable whenever the other clauses are satisfiable (except in the trivial case when $f(x) = 0$ for all x), because we don't need to include both x_j and $\bar{x}_j$ in the same term. Furthermore they reduce the space of possibilities by a factor of $(3/4)^N$. So they seem worthwhile. (On the other hand, their effect on the running time appears to be negligible, at least with respect to Algorithm C in small-scale trials.)

cardinality constr
covering problem
BDD
generating function
don't-cares
evaluation of Boolean functions
truth tables

DNF
Tseytin encoding
author
Tseytin encoding
sideways sum

59. $f(x)\oplus\hat f(x) = x_2\bar x_3\bar x_6\bar x_{10}\bar x_{12}(\bar x_8\lor x_8(x_{13}\lor x_{15}))$ is a function of eight variables that has 7 solutions. Thus the probability is $7/256 = .02734375$.

60. A typical example with 32 given values of $f(x)$, chosen randomly, yielded

$$\hat f(x_1,\ldots,x_{20}) = x_4\bar x_7\bar x_{12} \lor \bar x_6 x_8\bar x_{11}x_{14}x_{20} \lor \bar x_9\bar x_{12}x_{18}\bar x_{19} \lor \bar x_{13}\bar x_{16}\bar x_{17}x_{19},$$

which of course is way off; it differs from $f(x)$ with probability $102752/2^{18} \approx .39$. With 64 training values, however,

$$\hat f(x_1,\ldots,x_{20}) = x_2\bar x_{13}\bar x_{15}x_{19} \lor \bar x_3\bar x_9\bar x_{19}\bar x_{20} \lor \bar x_6\bar x_{10}\bar x_{12} \lor \bar x_8 x_{10}\bar x_{12}$$

comes closer, disagreeing only with probability $404/2^{11} \approx .197$.

61. We can add 24 clauses $(p_{a,1}\lor q_{a,1}\lor p_{a,2}\lor \bar q_{a,2}\lor p_{a,3}\lor \bar q_{a,3}\lor\cdots\lor p_{b,1}\lor q_{b,1}\lor\cdots\lor p_{c,1}\lor q_{c,1}\lor\cdots\lor p_{d,1}\lor q_{d,1}\lor\cdots\lor \bar p_{d,10}\lor q_{d,10}\lor\cdots\lor p_{d,20}\lor q_{d,20})$, one for each permutation $abcd$ of $\{1,2,3,4\}$; the resulting clauses are satisfiable only by other functions $f(x)$.

But the situation is more complicated in larger examples, because a function can have many equivalent representations as a short DNF. A general scheme, to decide whether the function described by a particular setting $p'_{i,j}$ and $q'_{i,j}$ of the ps and qs is unique, would be to add more complicated clauses, which state that $p_{i,j}$ and $q_{i,j}$ give a different solution. Those clauses can be generated by the Tseytin encoding of

$$\bigvee_{i=1}^{M}\bigwedge_{j=1}^{N}\bigl((\bar p_{i,j}\land\bar x_j)\lor(\bar q_{i,j}\land x_j)\bigr) \;\oplus\; \bigvee_{i=1}^{M}\bigwedge_{j=1}^{N}\bigl((\bar p'_{i,j}\land\bar x_j)\lor(\bar q'_{i,j}\land x_j)\bigr).$$

62. Preliminary experiments by the author, with $N = 20$ and $p = 1/8$, seem to indicate that more data points are needed to get convergence by this method, but the SAT solver tends to run about 10 times faster. Thus, locally biased data points appear to be preferable unless the cost of observing the hidden function is relatively large.

Incidentally, the chance that $x^{(k)} = x^{(k-1)}$ was relatively high in these experiments ($(7/8)^{20} \approx .069$); so cases with $y^{(k)} = 0$ were bypassed.

63. With Tseytin encoding (24), it's easy to construct $6r+2n-1$ clauses in $2r+2n-1$ variables that are satisfiable if and only if α fails to sort the binary sequence $x_1\ldots x_n$. For example, the clauses when $\alpha = [1{:}2][3{:}4][1{:}3][2{:}4][2{:}3]$ are $(x_1\lor\bar l_1)\land(x_2\lor\bar l_1)\land(\bar x_1\lor\bar x_2\lor l_1)\land(\bar x_1\lor h_1)\land(\bar x_2\lor h_1)\land(x_1\lor x_2\lor\bar h_1)\land\cdots\land(l_4\lor\bar l_5)\land(h_3\lor\bar l_5)\land(\bar l_4\lor\bar h_3\lor l_5)\land(\bar l_4\lor h_5)\land(\bar h_3\lor h_5)\land(l_4\lor h_3\lor\bar h_5)\land(g_1\lor g_2\lor g_3)\land(\bar g_1\lor l_3)\land(\bar g_1\lor\bar l_5)\land(\bar g_2\lor l_5)\land(\bar g_2\lor\bar h_5)\land(\bar g_3\lor h_5)\land(\bar g_3\lor\bar h_4)$. They're unsatisfiable, so α always sorts properly.

64. Here we reverse the policy of the previous answer, and construct clauses that are *satisfiable* when they describe a sorting network: Let the variable $C^t_{i,j}$ stand for the existence of comparator $[i{:}j]$ at time t, for $1\le i<j\le n$ and $1\le t\le T$. Also, adapting (20) and (21), let variables $B^t_{j,k}$ be defined for $1\le j\le n-2$ and $1\le k\le n$, with clauses

$$(\overline{B}^t_{2j,k}\lor\overline{B}^t_{2j+1,k})\land(\overline{B}^t_{2j,k}\lor B^t_{j,k})\land(\overline{B}^t_{2j+1,k}\lor B^t_{j,k})\land(B^t_{2j,k}\lor B^t_{2j+1,k}\lor\overline{B}^t_{j,k}); \qquad (*)$$

in this formula we substitute $\{C^t_{1,k},\ldots,C^t_{k-1,k},C^t_{k,k+1},\ldots,C^t_{k,n}\}$ for the $n-1$ "leaf nodes" $\{B^t_{n-1,k},\ldots,B^t_{2n-3,k}\}$. These clauses prohibit comparators from clashing at time t, and they make $B^t_{1,k}$ false if and only if line k remains unused.

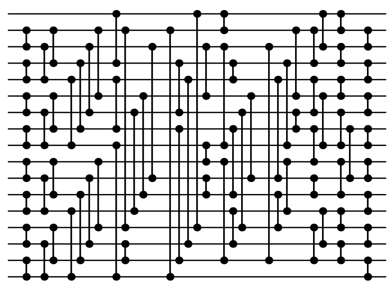
If $x = x_1\ldots x_n$ is any binary vector, let $y_1\ldots y_n$ be the result of sorting x (so that $(y_1\ldots y_n)_2 = 2^{\nu x}-1$). The following clauses $F(x)$ encode the fact that comparators $C^t_{i,j}$ transform $x\mapsto y$: $(\overline{C}^t_{i,j}\lor\overline{V}^t_{x,i}\lor V^{t-1}_{x,i})\land(\overline{C}^t_{i,j}\lor\overline{V}^t_{x,i}\lor V^{t-1}_{x,j})\land(\overline{C}^t_{i,j}\lor V^t_{x,i}\lor\overline{V}^{t-1}_{x,i}\lor\overline{V}^{t-1}_{x,j})\land(\overline{C}^t_{i,j}\lor\overline{V}^t_{x,j}\lor V^{t-1}_{x,i}\lor V^{t-1}_{x,j})\land(\overline{C}^t_{i,j}\lor V^t_{x,j}\lor\overline{V}^{t-1}_{x,i})\land(\overline{C}^t_{i,j}\lor V^t_{x,j}\lor\overline{V}^{t-1}_{x,j})\land(B^t_{1,i}\lor\overline{V}^t_{x,i}\lor V^{t-1}_{x,i})\land$

$(B^t_{1,i} \vee V^t_{x,i} \vee \overline{V}^{t-1}_{x,i})$, for $1 \le i < j \le n$ and $1 \le t \le T$; here we substitute x_j for $V^0_{x,j}$ and also substitute y_j for $V^T_{x,j}$, thereby simplifying the boundary conditions.

Furthermore, we can remove all variables $V^t_{x,i}$ when x has i leading 0s and $V^t_{x,j}$ when x has j trailing 1s, replacing them by 0 and 1 respectively and simplifying further.

Finally, given any sequence $\alpha = [i_1\,{:}\,j_1]\ldots[i_r\,{:}\,j_r]$ of initial comparators, T further parallel stages will yield a sorting network if and only if the clauses $(*)$, together with $\bigwedge_x F(x)$ over all x producible by α, are simultaneously satisfiable.

Setting $n = 9$, $\alpha = [1{:}6][2{:}7][3{:}8][4{:}9]$, and $T = 5$, we obtain 85768 clauses in 5175 variables, if we leave out the ten vectors x that are already sorted. Algorithm C finds them unsatisfiable after spending roughly 200 megamems; therefore $\hat{T}(9) > 6$. (Algorithm L fails spectacularly on these clauses, however.) Setting $T \leftarrow 6$ quickly yields $\hat{T}(9) \le 7$. D. Bundala and J. Závodný [*LNCS* **8370** (2014), 236–247] used this approach to prove in fact that $\hat{T}(11) = 8$ and $\hat{T}(13) = 9$. Then T. Ehlers and M. Müller extended it [arXiv:1410.2736 [cs.DS] (2014), 10 pages], to prove that $\hat{T}(17) = 10$, with the surprising optimum network shown here.



Bundala
Závodný
Ehlers
Müller
CNF
cardinality constraints
elegant

65. (a) The goal is to express the transition equation in CNF. There are $\binom{8}{4}$ clauses like $(\bar{x}' \vee \bar{x}_a \vee \bar{x}_b \vee \bar{x}_c \vee \bar{x}_d)$, one for each choice of four neighbors $\{a,b,c,d\} \subseteq \{\text{NW}, \text{N}, \ldots, \text{SE}\}$. Also $\binom{8}{7}$ clauses like $(\bar{x}' \vee x_a \vee \cdots \vee x_g)$, one for each choice of seven. Also $\binom{8}{6}$ like $(\bar{x}' \vee x \vee x_a \vee \cdots \vee x_f)$, for each choice of six. Also $\binom{8}{3}$ like $(x' \vee \bar{x}_a \vee \bar{x}_b \vee \bar{x}_c \vee x_d \vee \cdots \vee x_h)$, complementing just three. And finally $\binom{8}{2}$ like $(x' \vee \bar{x} \vee \bar{x}_a \vee \bar{x}_b \vee x_c \vee \cdots \vee x_g)$, complementing just two and omitting any one of the others. Altogether $70 + 8 + 28 + 56 + 28 = 190$ clauses of average length $(70\cdot 5 + 8\cdot 8 + 28\cdot 8 + 56\cdot 9 + 28\cdot 9)/190 \approx 7.34$.

(b) Here we let $x = x_{ij}$, $x_{\text{NW}} = x_{(i-1)(j-1)}, \ldots, x_{\text{SE}} = x_{(i+1)(j+1)}$, $x' = x'_{ij}$. There are seven classes of auxiliary variables $a^{ij}_k, \ldots, g^{ij}_k$, each of which has two children; the meaning is that the sum of the descendants is $\ge k$. We have $k \in \{2,3,4\}$ for the a variables, $k \in \{1,2,3,4\}$ for the b and c variables, and $k \in \{1,2\}$ for d, e, f, g.

The children of a^{ij} are $b^{(i|1)j}$ and c^{ij}. The children of b^{ij} are $d^{i(j-(j\&2))}$ and $e^{i(j+(j\&2))}$. The children of c^{ij} are $f^{i'j'}$ and g^{ij}, where $i' = i+2$ and $j' = (j-1)\,|\,1$ if i is odd, otherwise $i' = i$ and $j' = j - (j\&1)$. The children of d^{ij} are $x_{(i-1)(j+1)}$ and $x_{i(j+1)}$. The children of e^{ij} are $x_{(i-1)(j-1)}$ and $x_{i(j-1)}$. The children of f^{ij} are $x_{(i-1)j}$ and $x_{(i-1)(j+1)}$. Finally, the children of g^{ij} are $x_{i'j}$ and $x_{i''j''}$, where $i' = i+1-((i\&1)\ll 1)$; and $(i'',j'') = (i+1, j\oplus 1)$ if i is odd, otherwise $(i'',j'') = (i-1, j-1+((j\&1)\ll 1))$. (OK — this isn't elegant. But hey, it works!)

If the children of p are q and r, the clauses that define p_k are $(p_k \vee \bar{q}_{k'} \vee \bar{r}_{k''})$ for $k' + k'' = k$ and $(\bar{p}_k \vee q_{k'} \vee r_{k''})$ for $k' + k'' = k+1$. In these clauses we omit $\bar{q}_0$ or $\bar{r}_0$; we also omit q_m or r_m when q or r has fewer than m descendants.

For example, these rules define d^{35}_1 and d^{35}_2 by the following six clauses:

$$(d^{35}_1 \vee \bar{x}_{26}),\ \ (d^{35}_1 \vee \bar{x}_{36}),\ \ (d^{35}_2 \vee \bar{x}_{26} \vee \bar{x}_{36}),\ \ (\bar{d}^{35}_1 \vee x_{26} \vee x_{36}),\ \ (\bar{d}^{35}_2 \vee x_{26}),\ \ (\bar{d}^{35}_2 \vee x_{36}).$$

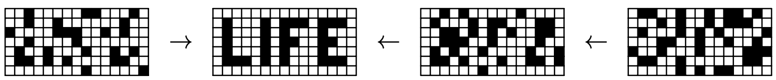
The variables b^{ij}_k are defined only when i is odd; d^{ij}_k and e^{ij}_k only when i is odd and $j \bmod 4 < 2$; f^{ij}_k only when $i+j$ is even. Thus the total number of auxiliary variables per cell (i,j), ignoring small corrections at boundary points, is $3 + 4/2 + 4 + 2/4 + 2/4 + 2/2 + 2 = 13$ of types a through g, not 19, because of the sharing; and the total number of clauses per cell to define them is $21 + 16/2 + 16 + 6/4 + 6/4 + 6/2 + 6 = 57$, not 77.

Finally we define x'_{ij} from a_2^{ij}, a_3^{ij}, a_4^{ij}, by means of six clauses

$$(\bar{x}'_{ij} \vee \bar{a}_4^{ij}),\ \ (\bar{x}'_{ij} \vee a_2^{ij}),\ \ (\bar{x}'_{ij} \vee x_{ij} \vee a_3^{ij}),\ \ (x'_{ij} \vee a_4^{ij} \vee \bar{a}_3^{ij}),\ \ (x'_{ij} \vee \bar{x}_{ij} \vee \bar{y}_{ij}),\ \ (y_{ij} \vee a_4^{ij} \vee \bar{a}_2^{ij}),$$

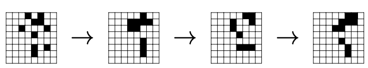
where y_{ij} is another auxiliary variable (introduced only to avoid clauses of size 4).

66. All solutions to (a) can be characterized by a BDD of 8852 nodes, from which we can obtain the generating function $38z^{28} + 550z^{29} + \cdots + 150z^{41}$ that enumerates them (with a total computation time of only 150 megamems or so). Part (b), however, is best suited to SAT, and X_0 must have at least 38 live cells. Typical answers are



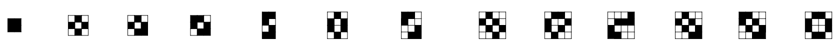
67. Either  or  at lower left will produce the X_0 of (37) at time 1. But length 22 is impossible: With $r = 4$ we can verify that all the live cells in X_4 lie in some 3×3 subarray. Then with $r = 22$ we need to rule out only $(\binom{9}{3} + \binom{9}{4} + \binom{9}{5}) \times 6 = 2016$ possibilities, one for each viable X_4 within each essentially different 3×3 subarray.

68. The author believes that $r = 12$ is impossible, but his SAT solvers have not yet been able to verify this conjecture. Certainly $r = 11$ is achievable, because we can continue with the text's fifth example after prepending



69. Since only 8548 essentially different 4×4 bitmaps are possible (see Section 7.2.3), an exhaustive enumeration is no sweat. The small stable patterns arise frequently, so they've all been named:

(a) block tub boat ship snake beehive carrier barge loaf eater long boat long ship pond



(b) blinker clock toad beacon



(A glider is also considered to be stable, although it's not an oscillator.)

70. (a) A cell with three live neighbors in the stator will stay alive.

(b) A $4 \times n$ board doesn't work; Fig. A–6 shows the 5×8 examples.

(c) Again, the smallest-weight solutions with smallest rectangles are shown in Fig. A–6. Oscillators with these rotors are plentiful on larger boards; the first examples of each kind were found respectively by Richard Schroeppel (1970), David Buckingham (1972), Robert Wainwright (1985).

71. Let the variables $X_t = x_{ijt}$ characterize the configuration at time t, and suppose we require $X_r = X_0$. There are $q = 8r$ automorphisms σ that take $X_t \mapsto X_{(t+p) \bmod r}\tau$, where $0 \le p < r$ and τ is one of the eight symmetries of a square grid.

Any global permutation of the $N = n^2 r$ variables leads via Theorem E to a canonical form, where we require the solution to be lexicographically less than or equal to the $q - 1$ solutions that are equivalent to it under automorphisms.

Such lexicographic tests can be enforced by introducing $(q-1)(3N-2)$ new clauses of length ≤ 3, as in (169) — and often greatly simplified using Corollary E.

These additional clauses can significantly speed up a proof of unsatisfiability. On the other hand they can also slow down the search, if a problem has abundant solutions.

BDD
author
stable
Schroeppel
Buckingham
Wainwright
automorphisms

In practice it's usually better to insist only on solutions that are *partially* canonical, by using only some of the automorphisms and by requiring lexicographic order only on some of the variables.

72. (a) The two 7 × 7s, shown in Fig. A–6, were found by R. Wainwright (trice tongs, 1972) and A. Flammenkamp (jam, 1988).

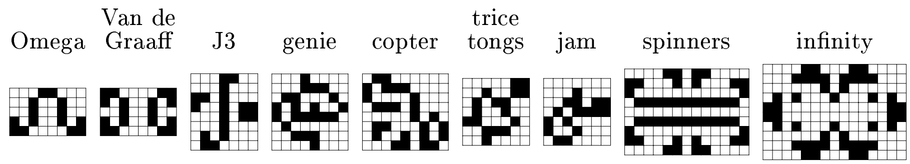
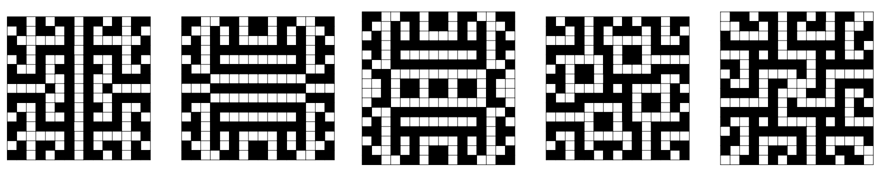


Fig. A–6. Noteworthy minimal oscillators of periods 2 and 3.

(b) Here the smallest examples are 9×13 and 10×15; the former has four L-rotors surrounding long stable lines. Readers will also enjoy discovering 10×10 and 13×13 instances that have full eightfold symmetry. (When encoding such symmetrical problems by using exercise 65(b), we need only compute the transitions between variables x_{tij} for $1 \le i \le \lceil m/2\rceil$ and $1 \le j \le \lceil n/2\rceil$; every other variable is identical to one of these. However, the auxiliary variables $a^{ij}, \ldots, g^{ij}$ shouldn't be coalesced in this way.)

(c,d) Champion heavyweights have small rotors. What a cool four-way snake dance!

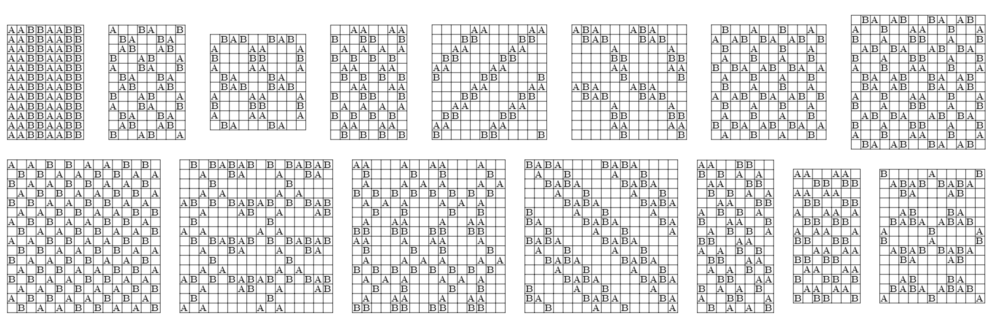


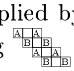
$120/225 \approx .53$ $130/240 \approx .54$ $132/256 \approx .52$ $120/225 \approx .53$ $136/256 \approx .53$

73. (a) They don't have three A neighbors; and they don't have three B neighbors.

(b) Two examples appear in Fig. A–7, where they are packed as snugly as possible into a 12×15 box. This pattern, found by R. W. Gosper about 1971, is called the *phoenix*, since its living cells repeatedly die and rise again. It is the smallest mobile flipflop; the same idea yields the next smallest (also seen in Fig. A–7), which is 10×12.

(c) The nonblank one comes from a 1×4 torus; the checkerboard from an 8×8. Here are some amazing $m \times n$ ways to satisfy the constraints for small m and n:



Notice that infinite one-dimensional examples are implied by several of these motifs; the checkerboard, in fact, can be fabricated by placing  diagonals together.

Wainwright
Flammenkamp
eightfold symmetry
encoding
Gosper
phoenix
quilt patterns

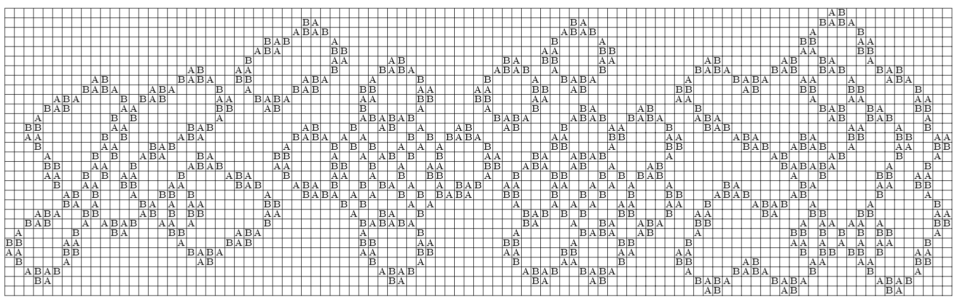


Fig. A–7. Mobile flipflops: An ideal way to tile the floor of a workspace for hackers.

74. Call a cell *tainted* if it is A with more than one A neighbor or B with more than one B neighbor. Consider the topmost row with a tainted cell, and the leftmost tainted cell in that row. We can assume that this cell is an A, and that its neighbors are S, T, U, V, W, X, Y, Z in the pattern STU/VAW/XYZ. Three of those eight neighbors are type B, and at least four are type A; several cases need to be considered.

Case 1: W = X = Y = Z = A. Then we must have S = U = V = B and T = 0 (blank), because S, T, U, V aren't tainted. The three left neighbors of V can't be type A, since V already has three A neighbors; nor can they be type B, since V isn't tainted. Hence the tainted X, which must have two B neighbors in the three cells below it, cannot also have two or more A neighbors there.

Case 2: T = A or V = A. If, say, T = A then X = Y = Z = A, and neither V nor W can be type B.

Case 3: S $\neq$ A, U = A. Then W can't be type B, and S must be tainted.

Case 4: S = A, U $\neq$ A. At least one of W, X, Y, Z is B; at least three are A; so exactly three are A. The B can't be Y, which has four A neighbors. Nor can it be W or Z: That would force V to be blank, hence T = U = B; consequently W = A, Z = B. Since W is tainted, at least two of its right neighbors must be A, contradicting Z = B.

Thus X = B in Case 4. Either T or V is also B, while the other is blank; say T is blank. The three left neighbors of V cannot be A. So they must either all be B (tainting the cell left of S) or all blank. In the latter case the upper neighbors of T must be BBA in that order, since T is blank. But that taints the B above T. A symmetric argument applies if V is blank.

Case 5: S = U = A. Then W $\neq$ A, and at least two of $\{X, Y, Z\}$ are A. Now Y = Z = A forces T = V = X = B and W blank, tarnishing V.

Similarly, X = Y = A forces T = W = Z = B and V blank; this case is more difficult. The three lower neighbors of Y must be AAB, in that order, lest a B be surrounded by four A's. But then the left neighbors of X are BBB; hence so are the left neighbors of V, tarnishing the middle one.

Finally, therefore, Case 5 implies that X = Z = A. Either T, V, W, or Y is blank; the other three are B. The blank can't be T, since T's upper three neighbors can't be A. It can't be W or Y, since V and T aren't tainted. So T = W = Y = B and V is blank. The left neighbors of S cannot be A, because S isn't tainted. So the cell left of X must be A. Therefore X must have at least four A neighbors; but that's impossible, because Y already has three.

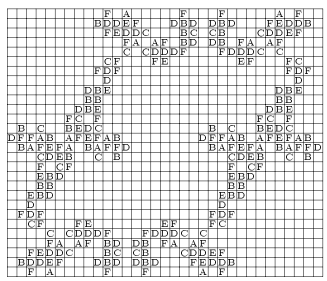
Summers
torus
Rokicki
still life
symeater
blocks
carriers
Buckingham
Silver
pixels
glider's symmetry

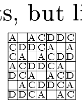
Diagonally adjacent A's are rare. (In fact, they cannot occur in rectangular grids of size less than 16 × 18.) But diligent readers will be able to spot them in Fig. A–7, which exhibits an astonishing variety of different motifs that are possible in large grids.

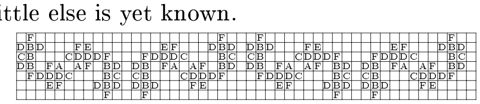
75. Let the cells alive at times $p-2$, $p-1$, p be of types X, Y, Z, and consider the topmost row in which a live cell ever appears. Without loss of generality, the leftmost cell in that row is type Z. The cell below that Z can't be of type Y, because that Y would have three X neighbors and four Y neighbors besides Z and the blank to its left.

Thus the picture must look like , where the three predecessors of Z and the topmost Y are filled in. But there's no room for the three predecessors of the topmost X.

76. The smallest known example, a 28×33 pattern found by Jason Summers in 2012, is illustrated here using the letters {F, A, B}, {B, C, D}, {D, E, F} for cells that are alive when $t \bmod 3 = 0, 1, 2$. His ingenious construction leads in particular to an infinite solution based on a 7×24 torus. An amazing infinite 7 × 7 toroidal pattern also exists, but little else is yet known.







77. If the first four cells in row 4 of X_0 (and of X_5) contain a, b, c, d, we need $a+b \neq 1$, $a+b+c \neq 1$, $b+c+d \neq 2$. In clause form this becomes $\bar a \vee b$, $a \vee \bar b$, $b \vee \bar c$, $\bar c \vee d$, $\bar b \vee c \vee \bar d$.

Similarly, let the last four elements of column 5 be (f, g, h, i); then we want $f+g+h \neq 2$, $g+h+i \neq 2$, $h+i \neq 2$. These conditions simplify to $\bar f \vee \bar g$, $\bar f \vee \bar h$, $\bar g \vee \bar\imath$, $\bar h \vee \bar\imath$.

78. The "9^2 phage" in Fig. A–8 is a minimal example.

79. (Solution by T. G. Rokicki.) A tremendous battle flares up, raging wildly on all fronts. When the dust finally settles at time 1900, 11 gliders are escaping the scene (1 going in the original NE direction, 3 going NW, 5 going SW, and 2 going SE), leaving behind 16 blocks, 1 tub, 2 loaves, 3 boats, 4 ships, 8 beehives, 1 pond, 15 blinkers, and 1 toad. (One should really watch this with a suitable applet.)

80. Paydirt is hit on 10 × 10 and 11 × 11 boards, with $X_8 = X_9$; see Fig. A–8. The minimal example, "symeater19," has a close relative, "symeater20," which consists simply of two blocks and two carriers, strategically placed. (The first of these, also called "eater 2," was discovered by D. Buckingham in the early 1970s; the other by S. Silver in 1998.) They both have the additional ability to eat the glider if it is moved one or two cells to the right of the position shown, or one cell to the left.

It is important to realize that the diagonal track of a glider does *not* pass through the corners of pixels, bisecting them; the axis of a glider's symmetry actually passes through the *midpoints* of pixel edges, thereby cutting off small triangles whose area is 1/8 of a full pixel. Consequently, any eater that is symmetric about a diagonal will eat gliders in two adjacent tracks. The two in Fig. A–8 are exceptional because they're quadruply effective. Furthermore symeater20 will eat from the opposite direction; and either of its carriers can be swapped to another position next to the blocks.

81. Two eaters make "ssymeater14" (Fig. A–8); and "ssymeater22" is narrower.

82. (a) If $X \to X'$, then $x'_{ij} = 1$ only if we have $\sum_{i'=i-1}^{i+1} \sum_{j'=j-1}^{j+1} x_{i'j'} \geq 3$.

(b) Use the same inequality, and induction on j.

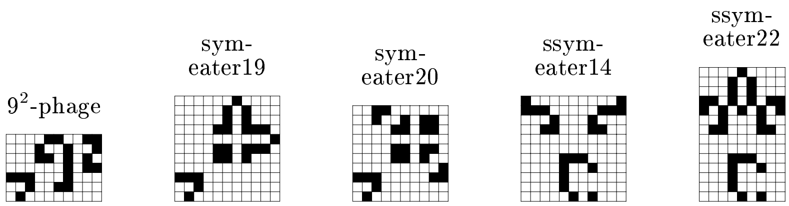
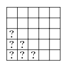
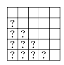
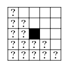


Fig. A–8. Various examples of minimal still lifes that eat gliders and spaceships.

(c) (Proof of the hint by John Conway, 1970.) In the transitions

X =  →  →  = X'',

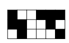
we must have  in the center of X'; hence we must have  at the lower left of X. But then the center of X' is .

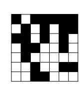
83. Work with $(2r+1-2t)\times(2r+1-2t)$ grids x_{tij} centered at cell (i_0,j_0), for $0\le t\le r=f(i_0,j_0)$; and assume that $x_{tij}=0$ whenever $f(i,j)>t$. For example, if $(i_0,j_0)=(1,2)$, only 14 of the x_{3ij} can be alive, namely when $(i,j)=(-2\,.\,.-1,2)$, $(-2\,.\,.\,0,1)$, $(-2\,.\,.\,1,0)$, $(-2\,.\,.\,2,-1)$. The case $(i_0,j_0)=(1,2)$ leads to 5031 readily satisfiable clauses on 1316 variables, including the unit clause x_{612}, when the state transitions are encoded as in answer 65; all but 106 of those variables are auxiliary.

84. (a) Use a glider, positioned properly with its tip at $(0,0)$.

(b) Similarly, a spaceship reaches these cells in the minimum possible time.

(c) Consider patterns A_n =  and B_n =  of width $2n+1$, illustrated here for $n=3$. Then B_j works when $j \bmod 4\in\{1,2\}$; A_j and B_{j-1} work when $j \bmod 4\in\{2,3\}$; A_{j-1} works when $j \bmod 4\in\{0,3\}$.

(d) The pattern  assembles a suitable glider at time 3.



(e) A SAT solver found the pattern shown here, which launches an appropriate spaceship (plus some construction debris that vanishes at $t=5$).

[It appears likely that $f^*(i,j)=f(i,j)$ for all i and j. But the best general result at present, based on space-filling constructions such as Tim Coe's "Max," is that $f^*(i,j)=f(i,j)+O(1)$. There's no known way to prove even the special cases that, say, $f^*(j,2j)=6j$ or that $f^*(-j,2j)=3j$ for all $j\ge 0$.]

85. (a) Let X be a 12×12 bitmap. We must show that the clauses $T(X,X')$ of exercise 65, together with 92 unary clauses x'_{23}, $\bar{x}'_{24}$, x'_{25}, … from the given pattern, are unsatisfiable. (The pattern is symmetrical; but Life's rules often produce symmetrical states from unsymmetrical ones.) Thus 2^{144-8} different conceivable predecessor states need to be ruled out. Fortunately Algorithm C needs fewer than 100 Mμ to do that.

(b) Most states have thousands of predecessors (see the following exercise); so Algorithm C can almost always find one in, say, 500 Kμ. Therefore one can prove, for example, that no 6×6 Gardens of Eden exist, by rapidly finding a predecessor for each of the 2^{36} patterns. (Only about $2^{36}/8$ patterns actually need to be tried, by symmetry.) Furthermore, if we run through those patterns in Gray code order, changing the polarity of just one assumed unary clause $\pm x'_{ij}$ at each step, the mechanism of Algorithm C goes even faster, because it tends to find nearby solutions to nearby problems. Thus thousands of patterns can be satisfied per second, and the task is feasible.

Conway
glider
spaceship
Coe
symmetrical
Mμ
Kμ
Gray code

Such an approach is out of the question for 10×10 bitmaps, because $2^{100} \gg 2^{36}$. But we *can* find all 10×10 Gardens of Eden for which there is 90°-rotational symmetry, by trying only about $2^{25}/2$ patterns, again using Gray code. Aha: Eight such patterns have no predecessor, and four of them correspond to the given orphan.

[See C. Hartman, M. J. H. Heule, K. Kwekkeboom, and A. Noels, *Electronic J. Combinatorics* **20**, 3 (August 2013), #P16. The existence of Gardens of Eden with respect to many kinds of cellular automata was first proved nonconstructively by E. F. Moore, *Proc. Symp. Applied Math.* **14** (1962), 17–33.]

86. The 80 cells outside the inner 8×8 can be chosen in $N = 11{,}984{,}516{,}506{,}952{,}898$ ways. (A BDD of size 53464 proves this.) So the answer is $N/2^{100-64} \approx 174{,}398$.

87. Instead of using subscripts t and $t+1$, we can write the transition clauses for $X \to X'$ in the form $(@ \lor \overline{\mathrm{A0}} \lor \mathrm{A0}')$, etc. Let Alice's states be $\{\alpha_1, \ldots, \alpha_p\}$ and let Bob's be $\{\beta_1, \ldots, \beta_q\}$. The clauses $(@ \lor \bar\alpha_i \lor \alpha_i')$ and $(\overline{@} \lor \bar\beta_i \lor \beta_i')$ say that your state doesn't change unless you are bumped. If state α corresponds to the command 'Maybe go to s', the clause $(\overline{@} \lor \bar\alpha \lor \alpha' \lor s')$ defines the next possible states after bumping. The analogous clause for 'Critical, go to s' or 'Set $v \leftarrow b$, go to s' is simply $(\overline{@} \lor \bar\alpha \lor s')$; and the latter also generates the clause $(\overline{@} \lor \bar\alpha \lor v')$ if $b = 1$, $(\overline{@} \lor \bar\alpha \lor \bar v')$ if $b = 0$. The command 'If v go to s_1, else to s_0' generates $(\overline{@} \lor \bar\alpha \lor \bar v \lor s_1') \land (\overline{@} \lor \bar\alpha \lor v \lor s_0')$. And for each variable v, if the states whose commands set v are $\alpha_{i_1}, \ldots, \alpha_{i_h}$, the clauses

$$(\overline{@} \lor v \lor \alpha_{i_1} \lor \cdots \lor \alpha_{i_h} \lor \bar v') \;\land\; (\overline{@} \lor \bar v \lor \alpha_{i_1} \lor \cdots \lor \alpha_{i_h} \lor v')$$

encode the fact that v isn't changed by other commands.

Bob's program generates similar clauses — but they use @, not $\overline{@}$, and β, not α.

Incidentally, when other protocols are considered in place of (40), the initial state X_0 analogous to (41) is constructed by putting Alice and Bob into their smallest possible states, and by setting all shared variables to 0.

88. For example, let all variables be false except $\mathrm{A0}_0$, $\mathrm{B0}_0$, $@_0$, $\mathrm{A1}_1$, $\mathrm{B0}_1$, $\mathrm{A1}_2$, $\mathrm{B1}_2$, $\mathrm{A1}_3$, $\mathrm{B2}_3$, $@_3$, $\mathrm{A2}_4$, $\mathrm{B2}_4$, $@_4$, $\mathrm{A3}_5$, $\mathrm{B2}_5$, l_5, $\mathrm{A3}_6$, $\mathrm{B3}_6$, l_6.

89. No; we can find a counterexample to the corresponding clauses as in the previous exercise: $\mathrm{A0}_0$, $\mathrm{B0}_0$, $\mathrm{A0}_1$, $\mathrm{B1}_1$, $\mathrm{A0}_2$, $\mathrm{B2}_2$, b_2, $@_2$, $\mathrm{A1}_3$, $\mathrm{B2}_3$, b_3, $\mathrm{A1}_4$, $\mathrm{B3}_4$, b_4, $\mathrm{A1}_5$, $\mathrm{B4}_5$, b_5, $@_5$, $\mathrm{A2}_6$, $\mathrm{B4}_6$, a_6, b_6, $@_6$, $\mathrm{A5}_7$, $\mathrm{B4}_7$, a_7, b_7, $\mathrm{A5}_8$, $\mathrm{B2}_8$, a_8, b_8, l_8, $\mathrm{A5}_9$, $\mathrm{B5}_9$, a_9, b_9, l_9.

(This protocol was the author's original introduction to the fascinating problem of mutual exclusion [see *CACM* **9** (1966), 321–322, 878], about which Dijkstra had said "Quite a collection of trial solutions have been shown to be incorrect.")

90. Alice starves in (43) with $p = 1$ and $r = 3$ in (47), if she moves to A1 and then Bob remains in B0 whenever he is bumped. The $\mathrm{A2} \land \mathrm{B2}$ deadlock mentioned in the text for (45) corresponds to (47) with $p = 4$ and $r = 6$. And in (46), successive moves to B1, (B2, A1, A2, B3, B1, A4, A5, A0)$^\infty$ will starve poor Bob.

91. A cycle (47) with no maybe/critical states for Alice can certainly starve her. Conversely, given (i), (ii), (iii), suppose Alice is in no maybe/critical state when $t \ge t_0$; and let $t_0 < t_1 < t_2 < \cdots$ be times with $@_{t_i} = 1$ but with $@_t = 0$ for at least one t between t_i and t_{i+1}. Then we must have $X_{t_i} = X_{t_j}$ for some $i < j$, because the number of states is finite. Hence there's a starvation cycle with $p = t_i$ and $r = t_j$.

92. For $0 \le i < j \le r$ we want clauses that encode the condition $X_i \ne X_j$. Introduce new variables σ_{ij} for each state σ of Alice or Bob, and v_{ij} for each shared variable v. Assert that at least one of these new variables is true. (For the protocol (40) this clause

90°-rotational symmetry
Hartman
Heule
Kwekkeboom
Noels
cellular automata
nonconstructively
Moore
BDD
transition clauses
initial state
Knuth
Dijkstra
encode

would be $(\mathrm{A0}_{ij} \vee \cdots \vee \mathrm{A4}_{ij} \vee \mathrm{B0}_{ij} \vee \cdots \vee \mathrm{B4}_{ij} \vee l_{ij})$.) Also assert the binary clauses $(\bar\sigma_{ij} \vee \sigma_i) \wedge (\bar\sigma_{ij} \vee \bar\sigma_j)$ for each σ, and the ternary clauses $(\bar v_{ij} \vee v_i \vee v_j) \wedge (\bar v_{ij} \vee \bar v_i \vee \bar v_j)$ for each v.

The transition clauses can also be streamlined, because we needn't allow cases where $X_{t+1} = X_t$. Thus, for example, we can omit $\mathrm{B0}_{t+1}$ from the clause $(@_t \vee \overline{\mathrm{B0}_t} \vee \mathrm{B0}_{t+1} \vee \mathrm{B1}_{t+1})$ of (42); and we can omit the clause $(@_t \vee \overline{\mathrm{B1}_t} \vee \bar l_t \vee \mathrm{B1}_{t+1})$ entirely.

[If r is large, encodings with $O(r(\log r)^2)$ clauses are possible via sorting networks, as suggested by D. Kroening and O. Strichman, *LNCS* **2575** (2003), 298–309. The most practical scheme, however, seems to be to add the ij constraints one by one as needed; see N. Eén and N. Sörensson, *Electronic Notes in Theoretical Computer Science* **89** (2003), 543–560.]

sorting networks
Kroening
Strichman
Eén
Sörensson
k-induction
induction
Sheeran
Singh
Stålmarck
longest simple path

93. For the Φ in (50), for example, we can use $(x_1 \vee x_2 \vee \cdots \vee x_{16}) \wedge (\bar x_1 \vee \overline{\mathrm{A0}'}) \wedge \cdots \wedge (\bar x_1 \vee \overline{\mathrm{A6}'}) \wedge (\bar x_2 \vee \overline{\mathrm{B0}'}) \wedge \cdots \wedge (\bar x_2 \vee \overline{\mathrm{B6}'}) \wedge (\bar x_3 \vee \mathrm{A0}') \wedge (\bar x_3 \vee a') \wedge \cdots \wedge (\bar x_{16} \vee \mathrm{B6}') \wedge (\bar x_{16} \vee \bar b')$.

94. $(X \to X' \to \cdots \to X^{(r)}) \wedge \Phi(X) \wedge \Phi(X') \wedge \cdots \wedge \Phi(X^{(r-1)}) \wedge \neg\Phi(X^{(r)})$. [This important technique is called "k-induction"; see Mary Sheeran, Satnam Singh, and Gunnar Stålmarck, *LNCS* **1954** (2000), 108–125. One can, for example, add the clause $(\overline{\mathrm{A5}} \vee \overline{\mathrm{B5}})$ to (50) and prove the resulting formula Φ by 3-induction.]

95. The critical steps have $a = b = 1$, by the invariants, so they have no predecessor.

96. The only predecessor of $\mathrm{A5}_2 \wedge \mathrm{B5}_2 \wedge a_2 \wedge b_2 \wedge \bar l_2$ is $\mathrm{A5}_1 \wedge \mathrm{B4}_1 \wedge a_1 \wedge b_1 \wedge \bar l_1$; and the only predecessor of *that* is $\mathrm{A5}_0 \wedge \mathrm{B3}_0 \wedge a_0 \wedge b_0 \wedge \bar l_0$. The case l_2 is similar.

But *without* the invariants, we could find arbitrarily long paths to $\mathrm{A5}_r \wedge \mathrm{B5}_r$. In fact the longest such *simple* path has $r = 33$: Starting with $\mathrm{A2}_0 \wedge \mathrm{B2}_0 \wedge \bar a_0 \wedge \bar b_0 \wedge l_0$, we could successively bump Alice and Bob into states A3, A5, A6, A0, A1, A2, A3, B3, B4, A5, B3, A6, B4, A0, B3, A1, A2, A3, A5, A6, A0, A1, A2, B4, A3, A5, A6, A0, B5, A1, A2, A3, A5, never repeating a previous state. (Of course all of these states are unreachable from the real X_0, because none of them satisfy Φ.)

97. No. Removing each person's final step in a path to $\mathrm{A6} \wedge \mathrm{B6}$ gives a path to $\mathrm{A5} \wedge \mathrm{B5}$.

98. (a) Suppose $X_0 \to \cdots \to X_r = X_0$ is impure and $X_i = X_j$ for some $0 \le i < j < r$. We may assume that $i = 0$. If either of the two cycles $X_0 \to \cdots \to X_j = X_0$ or $X_j \to \cdots \to X_r = X_j$ is impure, it is shorter.

(b) In those states she would have had to be previously in A0 or A5.

(c) Generate clauses $(\bar g_0)$, $(\bar g_t \vee g_{t-1} \vee @_{t-1})$, $(\bar h_0)$, $(\bar h_t \vee h_{t-1} \vee \overline{@_{t-1}})$, $(\bar f_t \vee g_t)$, $(\bar f_t \vee h_t)$, $(\bar f_t \vee \alpha_0 \vee \bar\alpha_t)$, $(\bar f_t \vee \bar\alpha_0 \vee \alpha_t)$, $(\bar f_t \vee v_0 \vee \bar v_t)$, $(\bar f_t \vee \bar v_0 \vee v_t)$, for $1 \le t \le r$; and $(f_1 \vee f_2 \vee \cdots \vee f_r)$. Here v runs through all shared variables, and α runs through all states that can occur in a starvation cycle. (For example, Alice's states with respect to protocol (49) would be restricted to A3 and A4, but Bob's are unrestricted.)

(d) With exercise 92 we can determine that the longest simple path, using only states that can occur in a starvation cycle for (49), is 15. And the clauses of (c) are unsatisfiable when $r = 15$ and invariant (50) is used. Thus the only possible starvation cycle is made from two simple pure cycles; and those are easy to rule out.

99. Invariant assertions define the values of a and b at each state. Hence mutual exclusion follows as in exercise 95. For starvation-freedom, we can exclude states A0, A6, A7, A8 from any cycle that starves Alice. But we need also to show that the state $\mathrm{A5}_t \wedge \mathrm{B0}_t \wedge l_t$ is impossible; otherwise she could starve while Bob is maybe-ing. For that purpose we can add $\neg\big((\mathrm{A6} \vee \mathrm{A7} \vee \mathrm{A8}) \wedge (\mathrm{B6} \vee \mathrm{B7} \vee \mathrm{B8})\big) \wedge \neg(\mathrm{A8} \wedge \bar l) \wedge \neg(\mathrm{B8} \wedge l) \wedge \neg\big((\mathrm{A3} \vee \mathrm{A4} \vee \mathrm{A5}) \wedge \mathrm{B0} \wedge l\big) \wedge \neg\big(\mathrm{A0} \wedge (\mathrm{B3} \vee \mathrm{B4} \vee \mathrm{B5}) \wedge \bar l\big)$ to the invariant $\Phi(X)$. The longest simple path through allowable states has length 42; and the clauses of exercise

98(c) are unsatisfiable when $r = 42$. Notice that Alice and Bob never compete when setting the common variable l, because states A7 and B7 cannot occur together.

(See Dijkstra's *Cooperating Sequential Processes*, cited in the text.)

100. Bob is starved by the moves B1, (A1, A2, A3, B2, A4, B3, A0, B4, B1)$^\infty$. But an argument similar to the previous answer shows that Alice cannot be.

[The protocol obviously provides mutual exclusion as in exercise 95. It was devised independently in the late 1970s by J. E. Burns and L. Lamport, as a special case of an N-player protocol using only N shared bits; see *JACM* **33** (1986), 337–339.]

101. The following solution is based on G. L. Peterson's elegant protocol for N processes in *ACM Transactions on Programming Languages and Systems* **5** (1983), 56–65:

A0. Maybe go to A1.	B0. Maybe go to B1.
A1. Set $a_1 \leftarrow 1$, go to A2.	B1. Set $b_1 \leftarrow 1$, go to B2.
A2. If b_2 go to A2, else to A3.	B2. If a_1 go to B2, else to B3.
A3. Set $a_2 \leftarrow 1$, go to A4.	B3. Set $b_2 \leftarrow 1$, go to B4.
A4. Set $a_1 \leftarrow 0$, go to A5.	B4. Set $b_1 \leftarrow 0$, go to B5.
A5. If b_1 go to A5, else to A6.	B5. If a_2 go to B5, else to B6.
A6. Set $a_1 \leftarrow 1$, go to A7.	B6. Set $b_1 \leftarrow 1$, go to B7.
A7. If b_1 go to A8, else to A9.	B7. If a_1 go to B8, else to B12.
A8. If b_2 go to A7, else to A9.	B8. If a_2 go to B9, else to B12.
A9. Critical, go to A10.	B9. Set $b_1 \leftarrow 0$, go to B10.
A10. Set $a_1 \leftarrow 0$, go to A11.	B10. If a_1 go to B11, else to B6.
A11. Set $a_2 \leftarrow 0$, go to A0.	B11. If a_2 go to B10, else to B6.
	B12. Critical, go to B13.
(Alice and Bob might need an	B13. Set $b_1 \leftarrow 0$, go to B14.
app to help them deal with this.)	B14. Set $b_2 \leftarrow 0$, go to B0.

102. The clauses for, say, 'B5. If a go to B6, else to B7.' should be $(@ \vee \overline{\mathrm{B5}} \vee \bar a \vee \alpha_1 \vee \cdots \vee \alpha_p \vee \mathrm{B6}') \wedge (@ \vee \overline{\mathrm{B5}} \vee a \vee \alpha_1 \vee \cdots \vee \alpha_p \vee \mathrm{B7}') \wedge (@ \vee \overline{\mathrm{B5}} \vee \mathrm{B6}' \vee \mathrm{B7}')$, where α_1, …, α_p are the states in which Alice sets a.

103. See, for example, any front cover of *SICOMP*, or of *SIAM Review* since 1970.

104. Assume that $m \le n$. The case $m = n$ is clearly impossible, because all four corners must be occupied. When m is odd and $n = m + k + 1$, put m bishops in the first and last columns, then k in the middle columns of the middle row. When m is even and $n = m + 2k + 1$, put m in the first and last columns, and two in the middle rows of columns $m/2 + 2j$ for $1 \le j \le k$. There's no solution when m and n are both even, because the maximum number of independent bishops of each color is $(m + n - 2)/2$. [R. Berghammer, *LNCS* **6663** (2011), 103–106.]

105. (a) We must have $(x_{ij}, x'_{ij}) = (1, 0)$ for t pairs ij, and $(0, 1)$ for t other pairs; otherwise $x_{ij} = x'_{ij}$. Hence there are 2^{mn-2t} solutions.

(b) Use $2mn$ variables y_{ij}, y'_{ij} for $1 \le i \le m$ and $1 \le j \le n$, with binary clauses $(\bar y_{ij} \vee \bar y'_{ij})$, together with $m + n + 2(m + n - 1)$ sets of cardinality constraints such as (20) and (21) to enforce the balance condition $\sum\{y_{ij} + \bar y'_{ij} \mid ij \in L\} = |L|$ for each row, column, and diagonal line L.

(c) $T(m, n) = 1$ when $\min(m, n) < 4$, because only the zero matrix qualifies in such cases. Other values can be enumerated by backtracking, if they are small enough. (The asymptotic behavior is unknown.)

n =	4	5	6	7	8
$T(4,n)$ =	3	7	17	35	77
$T(5,n)$ =	7	31	109	365	1367
$T(6,n)$ =	17	109	877	6315	47607
$T(7,n)$ =	35	365	6315	107637	1703883
$T(8,n)$ =	77	1367	47607	1703883	66291089

(d) Suppose $m \le n$. Any solution with nonzero top row, bottom row, left column, and right column has all entries zero except

Dijkstra
Burns
Lamport
Peterson
SIAM
Berghammer
cardinality constraints
backtracking

that $y_{1t} = -y_{t1} = y_{(m+1-t)1} = -y_{mt} = y_{m(n+1-t)} = -y_{(m+1-t)n} = y_{tn} = -y_{1(n+1-t)}$, for some t with $1 < t \le m/2$. So the answer is $2\sum_{k=3}^{m}\lfloor(k-1)/2\rfloor(m-k)(n-k)$, which simplifies to $q(q-1)(4q(n-q)-5n+2q+3+(m \bmod 2)(6n-8q-5))/3$ when $q = \lfloor m/2 \rfloor$.

Gerdes
unit clauses

[The answer in the case $(m,n) = (25,30)$ is 36080; hence a random 25×30 image will have an average of $36080/256 \approx 140.9$ tomographically equivalent "neighbors" that differ from it in exactly eight pixel positions. Figure 36 has five such neighbors, one of which is shown in answer 111 below.]

(e) We can make all entries nonzero except on the main diagonals (see below). This is optimum, because the diagonal lines for a_1, a_3, $\ldots$, a_{4n-1}, b_1, b_3, $\ldots$, b_{4n-1} must each contain a different 0. So the answer is $2n(n-1)$. (But the maximum for odd sized boards is unknown; for $n = (5,7,9)$ it turns out to be $(6,18,33)$.)

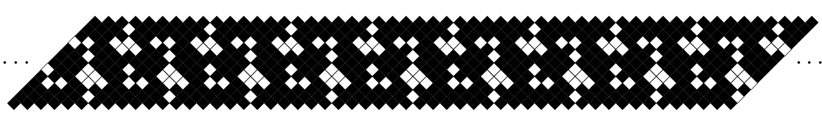
	0+++0−−−0		
0+++−−−0	−−−−+++0+	0++−−00	0++−−00
−0++−−0+	0−+−+−−++	−−++000	−−++000
−−0+−0++	++−++0−−−	0+−−00+	0+−0−0+
−−−00+++	−+−+−−0++	−00+0−+	−0−+00+
+++00−−−	+0−0+0+−−	+0−0+0−	+000+−−
++0−+0−−	−−+−−+++0	+0+0−0−	+0+−00−
+0−−++0−	++++−0−−−	0−−0++0	0−−0++0
0−−−+++0	0−0−−+++0		

(f) The smallest counterexamples are 7×7 (see above).

106. In an $m \times n$ problem we must have $0 \le r_i \le n$, $0 \le c_j \le m$, and $0 \le a_d, b_d \le \min\{d, m, n, m+n-d\}$. So the total number B of possibilities, assuming that $m \le n$, is $(n+1)^m(m+1)^n((m+1)!\,(m+1)^{n-m}m!)^2$, which is $\approx 3\cdot 10^{197}$ when $(m,n) = (25,30)$. Since $2^{750}/B \approx 2\cdot 10^{28}$, we conclude that a "random" 25×30 digital tomography problem usually has more than 10^{28} solutions. (Of course there are other constraints too; for example, the fact that $\sum r_i = \sum c_j = \sum a_d = \sum b_d$ reduces B by at least a factor of $(n+1)(m+1)^2$.)

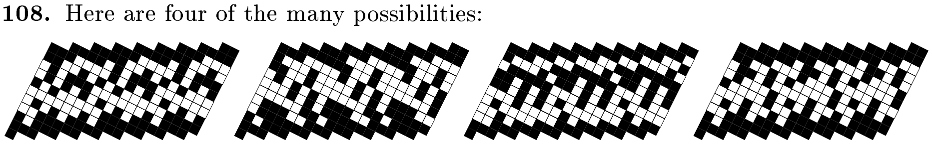
107. (a) $(r_1,\ldots,r_6) = (11,11,11,9,9,10)$; $(c_1,\ldots,c_{13}) = (6,5,6,2,4,4,6,5,4,2,6,5,6)$; $(a_1,\ldots,a_6) = (11,10,9,9,11,11)$; $(b_1,\ldots,b_{12}) = (6,1,6,5,7,5,6,2,6,5,7,5)$.

(b) There are two others, namely the following one and its left-right reversal:

. . .  . . .

[*Reference:* P. Gerdes, *Sipatsi* (Maputo: U. Pedagógica, 2009), page 62, pattern #122.]

108. Here are four of the many possibilities:



109. F1. [Initialize.] Find one solution $y_1 \ldots y_n$, or terminate if the problem is unsatisfiable. Then set $y_{n+1} \leftarrow 1$ and $d \leftarrow 0$.

F2. [Advance d.] Set d to the smallest $j > d$ such that $y_j = 1$.

F3. [Done?] If $d > n$, terminate with $y_1 \ldots y_n$ as the answer.

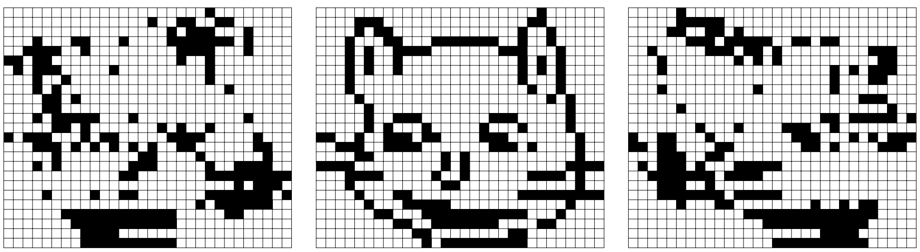
F4. [Try for smaller.] Try to find a solution with additional unit clauses to force $x_j = y_j$ for $1 \le j < d$ and $x_d = 0$. If successful, set $y_1 \ldots y_n \leftarrow x_1 \ldots x_n$. Return to F2. ▮

Even better is to incorporate a similar procedure into the solver itself; see exercise 275.

110. Algorithm B actually gives these directly:

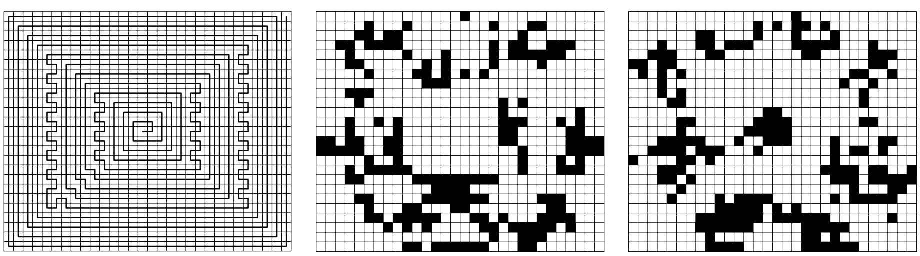
0011111110111011111100101111101111011111110111011011111111110010111110111101111110011101111111110111
1111111110111111110011001111110011110111111111010111111111011110111111100110011111011011110111111111

111. This family of problems appears to provide an excellent (though sometimes formidable) series of benchmark tests for SAT solvers. The suggested example has solutions



(a) colexicographically first; (b) minimally different; (c) colexicographically last;

and several of the entries in (a) were by no means easy. An even more difficult case arises if we base lexicographic order on a rook path that spirals out from the center (thus favoring solutions that are mostly 0 or mostly 1 in the middle):



(a) spiral rook path; (b) "spirographically" first; (c) "spirographically" last.

Here many of the entries have never yet been solved by a SAT solver, as of 2013, although again IP solvers have no great difficulty. In fact, the "lexicographic pure cutting plane" procedure of E. Balas, M. Fischetti, and A. Zanette [*Math. Programming* **A130** (2011), 153–176; **A135** (2012), 509–514] turns out to be particularly effective on such problems.

112. Reasonably tight upper and lower bounds would also be interesting.

113. Given an $N \times N \times N$ contingency problem with binary constraints $C_{JK} = X_{*JK}$, $R_{IK} = X_{I*K}$, $P_{IJ} = X_{IJ*}$, we can construct an equivalent $n \times n$ digital tomography problem with $n = N^2 + N^3 + N^4$ as follows: First construct a four-dimensional tensor $Y_{IJKL} = X_{(I \oplus L)JK}$, where $I \oplus L = 1 + (I + L - 1) \bmod N$, and notice that $Y_{*JKL} = Y_{IJK*} = X_{*JK}$, $Y_{I*KL} = X_{(I \oplus L)*K}$, $Y_{IJ*L} = X_{(I \oplus L)J*}$. Then define x_{ij} for $1 \le i, j \le n$ by the rule $x_{ij} = Y_{IJKL}$ when $i = I - N^2K + N^3L$, $j = NJ + N^2K + N^3L$, otherwise $x_{ij} = 0$. This rule makes sense; for if $1 \le I, I', J, J', K, K', L, L' \le N$ and $I - N^2K + N^3L = I' - N^2K' + N^3L'$ and $NJ + N^2K + N^3L = NJ' + N^2K' + N^3L'$, we have $I \equiv I'$ (modulo N); hence $I = I'$ and $K \equiv K'$; hence $K = K'$, $L = L'$, $J = J'$.

Under this correspondence the marginal sums are $r_i = Y_{I*KL}$ when $i = I - N^2K + N^3L$, $c_j = Y_{*JKL}$ when $j = NJ + N^2K + N^3L$, $a_d = Y_{IJ*L}$ when $d+1 = I + NJ + 2N^3L$, $b_d = Y_{IJK*}$ when $d - n = I - NJ - 2N^2K$, otherwise zero. [See S. Brunetti, A. Del Lungo, P. Gritzmann, and S. de Vries, *Theoretical Comp. Sci.* **406** (2008), 63–71.]

benchmark tests
rook path
spiral
cutting plane
Balas
Fischetti
Zanette
Brunetti
Del Lungo
Gritzmann
de Vries

symmetric function
polarities
smile
NP-complete
coNP-complete
Kaye
Scott
Stege
van Rooij
phoenix

114. (a) From $x_{7,23} + x_{7,24} = x_{7,23} + x_{7,24} + x_{7,25} = x_{7,24} + x_{7,25} = 1$ we deduce $x_{7,23} = x_{7,25} = 0$ and $x_{7,24} = 1$, revealing $n_{7,23} = n_{7,25} = 5$. Now $x_{6,23} + x_{6,24} = x_{6,24} + x_{6,25} = x_{4,24} + x_{5,24} + x_{6,24} + x_{6,25} = 1$; hence $x_{4,24} = x_{5,24} = 0$, revealing $n_{4,24} = n_{5,24} = 2$. So $x_{6,23} = x_{6,25} = 0$, and the rest is easy.

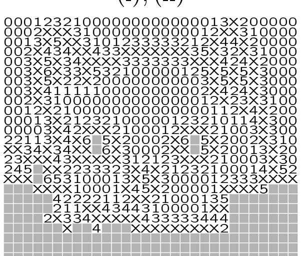
(b) Let $y_{i,j}$ mean "cell (i,j) has been probed safely, revealing $n_{i,j}$." Consider the clauses C obtained by appending $\bar{y}_{i,j}$ to each clause of the symmetric function $\bigl[\sum_{i'=i-1}^{i+1}\sum_{j'=j-1}^{j+1} x_{i',j'} = n_{i,j}\bigr]$, for all i,j with $x_{i,j} = 0$. Also include $(\bar{x}_{i,j} \vee \bar{y}_{i,j})$, as well as clauses for the symmetric function $S_N(x)$ if we're told the total number N of mines.

Given any subset F of mine-free cells, the clauses $C_F = C \wedge \bigwedge\{y_{i,j} \mid (i,j) \in F\}$ are satisfiable precisely by the configurations of mines that are consistent with the data $\{n_{i,j} \mid (i,j) \in F\}$. Therefore cell (i,j) is safe if and only if $C_F \wedge x_{i,j}$ is unsatisfiable.

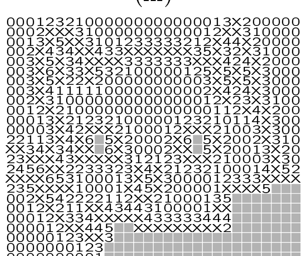
A simple modification of Algorithm C can be used to "grow" F until no further safe cells can be added: Given a solution to C_F for which neither $x_{i,j}$ nor $\bar{x}_{i,j}$ was obtained at root level (level 0), we can try to find a "flipped" solution by using the complemented value as the decision at level 1. Such a solution will be found if and only if the flipped value is consistent; otherwise the unflipped value will have been forced at level 0. By changing default polarities we can favor solutions that flip many variables at once. Whenever a literal $\bar{x}_{i,j}$ is newly deduced at root level, we can force $y_{i,j}$ to be true, thus adding (i,j) to F. We reach an impasse when a set of solutions has been obtained for C_F that covers both settings of every unforced $x_{i,j}$.

For problem (i) we start with $F = \{(1,1)\}$, etc. Case (iv) by itself uncovers only 56 cells in the lower right corner. The other results, each obtained in < 6 Gμ, are:

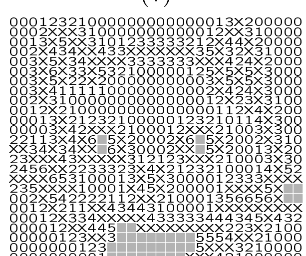
(i), (ii)



(iii)



(v)

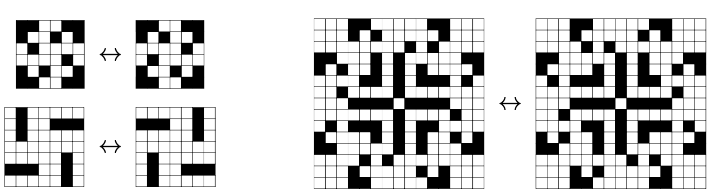


Notice that the Cheshire cat's famous smile defies logic and requires much guesswork!

[For aspects of Minesweeper that are NP-complete and coNP-complete, see Kaye, Scott, Stege, and van Rooij, *Math. Intelligencer* **22**, 2 (2000), 9–15; **33**, 4 (2011), 5–17.]

115. Several thousand runs of the algorithm in the previous exercise, given that the total number of mines is 10, indicate success probabilities $.490 \pm .007$, $.414 \pm .004$, $.279 \pm .003$, when the first guess is respectively in a corner, in the center of an edge, or in the center.

116. The smallest is the "clock" in answer 69(b). Other noteworthy possibilities are



as well as the "phoenix" in Fig. A–7.

partial backtracking
links dance
Pure literals

117. (a) Set $x_0 = x_{n+1} = 0$, and let (a, b, c) be respectively the number of occurrences of $(01, 10, 11)$ as a substring of $x_0x_1 \ldots x_{n+1}$. Then $a + c = b + c = \nu x$ and $c = \nu^{(2)}x$; hence $a = b = \nu x - \nu^{(2)}x$ is the number of runs.

(b) In this case the complete binary tree will have only $n-1$ leaves, corresponding to $\{x_1x_2, \ldots, x_{n-1}x_n\}$; therefore we want to replace n by $n-1$ in (20) and (21).

The clauses of (20) remain unchanged unless $t_k \le 3$. When $t_k = 2$ they become $(\bar{x}_{2k-n+1} \vee \bar{x}_{2k-n+2} \vee b_1^k) \wedge (\bar{x}_{2k-n+2} \vee \bar{x}_{2k-n+3} \vee b_1^k) \wedge (\bar{x}_{2k-n+1} \vee \bar{x}_{2k-n+2} \vee \bar{x}_{2k-n+3} \vee b_2^k)$. When $t_k = 3$ we have $2k = n-1$, and they become $(\bar{b}_1^{2k} \vee b_1^k) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee b_1^k) \wedge (\bar{b}_2^{2k} \vee b_2^k) \wedge (\bar{b}_1^{2k} \vee \bar{x}_1 \vee \bar{x}_2 \vee b_2^k) \wedge (\bar{b}_2^{2k} \vee \bar{x}_1 \vee \bar{x}_2 \vee b_3^k)$.

The clauses of (21) remain unchanged except in simple cases when $n \le 3$.

(c) Now the leaves represent $\overline{x_ix_{i+1}} = \bar{x}_i \vee \bar{x}_{i+1}$. So we change (20), when $t_k = 2$, to $(x_{2k-n+1} \vee b_1^k) \wedge (x_{2k-n+2} \vee b_1^k) \wedge (x_{2k-n+3} \vee b_1^k) \wedge (x_{2k-n+2} \vee b_2^k) \wedge (x_{2k-n+1} \vee x_{2k-n+3} \vee b_2^k)$. And there are eight clauses when $t_k = 3$: $(\bar{b}_1^{2k} \vee b_1^k) \wedge (x_1 \vee b_1^k) \wedge (x_2 \vee b_1^k) \wedge (\bar{b}_2^{2k} \vee b_2^k) \wedge (\bar{b}_1^{2k} \vee x_1 \vee b_2^k) \wedge (\bar{b}_1^{2k} \vee x_2 \vee b_2^k) \wedge (\bar{b}_2^{2k} \vee x_1 \vee b_3^k) \wedge (\bar{b}_2^{2k} \vee x_2 \vee b_3^k)$.

118. Let $p_{i,j}$ = [the pixel in row i and column j should be covered], and introduce variables $h_{i,j}$ when $p_{i,j} = p_{i,j+1} = 1$, $v_{i,j}$ when $p_{i,j} = p_{i+1,j} = 1$. The clauses are (i) $(h_{i,j} \vee h_{i,j-1} \vee v_{i,j} \vee v_{i-1,j})$, whenever $p_{i,j} = 1$, omitting variables that don't exist; (ii) $(\bar{h}_{i,j} \vee \bar{h}_{i,j-1})$, $(\bar{h}_{i,j} \vee \bar{v}_{i,j})$, $(\bar{h}_{i,j} \vee \bar{v}_{i-1,j})$, $(\bar{h}_{i,j-1} \vee \bar{v}_{i,j})$, $(\bar{h}_{i,j-1} \vee \bar{v}_{i-1,j})$, $(\bar{v}_{i,j} \vee \bar{v}_{i-1,j})$, whenever $p_{i,j} = 1$, omitting clauses whose variables don't both exist; and (iii) $(h_{i,j} \vee h_{i+1,j} \vee v_{i,j} \vee v_{i,j+1})$, whenever $p_{i,j} + p_{i,j+1} + p_{i+1,j} + p_{i+1,j+1} \ge 3$, omitting variables that don't exist. (The example has 10527 clauses in 2874 variables, but it's quickly solved.)

119. There's symmetry between l and $\bar{l}$, also between l and $10-l$; so we need consider only $l = (1, 2, 3, 4, 5)$, with respectively $(4, 4, 6, 6, 8)$ occurrences. The smallest result is $F \,|\, 5 = \{123, 234, 678, 789, 246, 468, 147, 369, \bar1\bar2\bar3, \bar2\bar3\bar4, \bar3\bar4, \bar4\bar6, \bar6\bar7, \bar6\bar7\bar8, \bar7\bar8\bar9, \bar1\bar3, \bar2\bar4\bar6, \bar3\bar7, \bar4\bar6\bar8, \bar7\bar9, \bar1\bar4\bar7, \bar2\bar8, \bar3\bar6\bar9, \bar1\bar9\}$.

120. True.

121. The main point of interest is that an empty clause is typically discovered in the *midst* of step A3; partial backtracking must be done when taking back the changes that were made before this interruption.

A3. [Remove $\bar{l}$.] Set $p \leftarrow \texttt{F}(\bar{l})$ (which is $\texttt{F}(l \oplus 1)$, see (57)). While $p \ge 2n+2$, set $j \leftarrow \texttt{C}(p)$, $i \leftarrow \texttt{SIZE}(j)$, and if $i > 1$ set $\texttt{SIZE}(j) \leftarrow i-1$, $p \leftarrow \texttt{F}(p)$. But if $i = 1$, interrupt that loop and set $p \leftarrow \texttt{B}(p)$; then while $p \ge 2n+2$, set $j \leftarrow \texttt{C}(p)$, $i \leftarrow \texttt{SIZE}(j)$, $\texttt{SIZE}(j) \leftarrow i+1$, $p \leftarrow \texttt{B}(p)$; and finally go to A5.

A4. [Deactivate l's clauses.] Set $p \leftarrow \texttt{F}(l)$. While $p \ge 2n+2$, set $j \leftarrow \texttt{C}(p)$, $i \leftarrow \texttt{START}(j)$, $p \leftarrow \texttt{F}(p)$, and for $i \le s < i + \texttt{SIZE}(j) - 1$ set $q \leftarrow \texttt{F}(s)$, $r \leftarrow \texttt{B}(s)$, $\texttt{B}(q) \leftarrow r$, $\texttt{F}(r) \leftarrow q$, and $\texttt{C}(\texttt{L}(s)) \leftarrow \texttt{C}(\texttt{L}(s)) - 1$. Then set $a \leftarrow a - \texttt{C}(l)$, $d \leftarrow d+1$, and return to A2.

A7. [Reactivate l's clauses.] Set $a \leftarrow a + \texttt{C}(l)$ and $p \leftarrow \texttt{B}(l)$. While $p \ge 2n+2$, set $j \leftarrow \texttt{C}(p)$, $i \leftarrow \texttt{START}(j)$, $p \leftarrow \texttt{B}(p)$, and for $i \le s < i + \texttt{SIZE}(j) - 1$ set $q \leftarrow \texttt{F}(s)$, $r \leftarrow \texttt{B}(s)$, $\texttt{B}(q) \leftarrow \texttt{F}(r) \leftarrow s$, and $\texttt{C}(\texttt{L}(s)) \leftarrow \texttt{C}(\texttt{L}(s)) + 1$. (The links dance a little here.)

A8. [Unremove $\bar{l}$.] Set $p \leftarrow \texttt{F}(\bar{l})$. While $p \ge 2n+2$, set $j \leftarrow \texttt{C}(p)$, $i \leftarrow \texttt{SIZE}(j)$, $\texttt{SIZE}(j) \leftarrow i+1$, $p \leftarrow \texttt{F}(p)$. Then go to A5. ▌

122. Pure literals are problematic when we want all solutions, so we don't take advantage of them here. Indeed, things get simpler; only the move codes 1 and 2 are needed.

A1*. [Initialize.] Set $d \leftarrow 1$.

A2*. [Visit or choose.] If $d > n$, visit the solution defined by $m_1 \ldots m_n$ and go to A6*. Otherwise set $l \leftarrow 2d + 1$ and $m_d \leftarrow 1$.

A3*. [Remove $\bar{l}$.] Delete $\bar{l}$ from all active clauses; but go to A5* if that would make a clause empty.

A4*. [Deactivate l's clauses.] Suppress all clauses that contain l. Then set $d \leftarrow d + 1$ and return to A2*.

A5*. [Try again.] If $m_d = 1$, set $m_d \leftarrow 2$, $l \leftarrow 2d$, and go to A3*.

A6*. [Backtrack.] Terminate if $d = 1$. Otherwise set $d \leftarrow d - 1$ and $l \leftarrow 2d + (m_d \,\&\, 1)$.

A7*. [Reactivate l's clauses.] Unsuppress all clauses that contain l.

A8*. [Unremove $\bar{l}$.] Reinstate $\bar{l}$ in all the active clauses that contain it. Then go back to A5*. ▮

It's no longer necessary to update the values $\texttt{C}(k)$ for $k < 2n+2$ in steps A4* and A7*.

123. For example, we might have

$$\begin{array}{rccccccccccccccccccccc} p = & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 \\ \texttt{L}(p) = & 3 & 9 & 7 & 8 & 7 & 5 & 6 & 5 & 3 & 4 & 3 & 8 & 2 & 8 & 6 & 9 & 6 & 4 & 7 & 4 & 2 \end{array}$$

and $\texttt{START}(j) = 21 - 3j$ for $0 \le j \le 7$; $W_2 = 3$, $W_3 = 7$, $W_4 = 4$, $W_5 = 0$, $W_6 = 5$, $W_7 = 1$, $W_8 = 6$, $W_9 = 2$. Also $\texttt{LINK}(j) = 0$ for $1 \le j \le 7$ in this case.

124. Set $j \leftarrow W_{\bar{l}}$. While $j \ne 0$, a literal other than $\bar{l}$ should be watched in clause j, so we do the following: Set $i \leftarrow \texttt{START}(j)$, $i' \leftarrow \texttt{START}(j-1)$, $j' \leftarrow \texttt{LINK}(j)$, $k \leftarrow i + 1$. While $k < i'$, set $l' \leftarrow \texttt{L}(k)$; if l' isn't false (that is, if $|l'| > d$ or $l' + m_{|l'|}$ is even, see (57)), set $\texttt{L}(i) \leftarrow l'$, $\texttt{L}(k) \leftarrow \bar{l}$, $\texttt{LINK}(j) \leftarrow W_{l'}$, $W_{l'} \leftarrow j$, $j \leftarrow j'$, and exit the loop on k; otherwise set $k \leftarrow k + 1$ and continue that loop. If k reaches i', however, we cannot stop watching $\bar{l}$; so we set $W_{\bar{l}} \leftarrow j$, exit the loop on j, and go on to step B5.

125. Change steps B2 and B4 to be like A2* and A4* in answer 122.

126. Starting with active ring (6 9 7 8), the unit clause 9 will be found (because 9 appears before 8); the clause $\bar{9}\bar{3}\bar{6}$ will become $\bar{6}\bar{3}\bar{9}$; the active ring will become (7 8 6).

127. Before: 11414545; after: 1142. (And then 11425, etc.)

128.

Active ring	$x_1x_2x_3x_4$	Units	Choice	Changed clauses
(1 2 3 4)	`- - - -`		$\bar{1}$	413
(2 3 4)	`0 - - -`		$\bar{2}$	$\bar{1}24$
(3 4)	`0 0 - -`	$\bar{3}$	$\bar{3}$	
(4)	`0 0 0 -`	$4, \bar{4}$	Backtrack	
(3 4)	`0 - - -`		2	$3\bar{2}\bar{1}, \bar{4}\bar{2}1$
(3 4)	`0 1 - -`	$\bar{4}$	$\bar{4}$	$314, \bar{3}42$
(3)	`0 1 - 0`	$3, \bar{3}$	Backtrack	
(4 3)	`- - - -`		1	$2\bar{1}4, \bar{4}\bar{1}\bar{3}$
(2 4 3)	`1 - - -`		2	
(4 3)	`1 1 - -`	3	3	$4\bar{3}\bar{2}, 2\bar{3}1$
(4)	`1 1 1 -`	$4, \bar{4}$	Backtrack	
(3 4)	`1 - - -`		$\bar{2}$	$\bar{3}21, 4\bar{1}2$
(3 4)	`1 0 - -`	4	4	$\bar{3}\bar{1}\bar{4}, 1\bar{2}\bar{4}, 3\bar{4}2$
(3)	`1 0 - 1`	$3, \bar{3}$	Backtrack	

129. Set $j \leftarrow W_l$, then do the following steps while $j \neq 0$: (i) Set $p \leftarrow \mathtt{START}(j) + 1$; (ii) if $p = \mathtt{START}(j-1)$, return 1; (iii) if $\mathtt{L}(p)$ is false (that is, if $x_{|\mathtt{L}(p)|} = \mathtt{L}(p)\ \&\ 1$), set $p \leftarrow p+1$ and repeat (ii); (iv) set $j \leftarrow \mathtt{LINK}(j)$. If j becomes zero, return 0.

130. Set $l \leftarrow 2k+b$, $j \leftarrow W_l$, $W_l \leftarrow 0$, and do the following steps while $j \neq 0$: (i) Set $j' \leftarrow \mathtt{LINK}(j)$, $i \leftarrow \mathtt{START}(j)$, $p \leftarrow i+1$; (ii) while $\mathtt{L}(p)$ is false, set $p \leftarrow p+1$ (see answer 129; this loop will end before $p = \mathtt{START}(j-1)$); (iii) set $l' \leftarrow \mathtt{L}(p)$, $\mathtt{L}(p) \leftarrow l$, $\mathtt{L}(i) \leftarrow l'$; (iv) set $p \leftarrow W_{l'}$ and $q \leftarrow W_{\bar{l}'}$, and go to (vi) if $p \neq 0$ or $q \neq 0$ or $x_{|l'|} \geq 0$; (v) if $t = 0$, set $t \leftarrow h \leftarrow |l'|$ and $\mathtt{NEXT}(t) \leftarrow h$, otherwise set $\mathtt{NEXT}(|l'|) \leftarrow h$, $h \leftarrow |l'|$, $\mathtt{NEXT}(t) \leftarrow h$ (thus inserting $|l'| = l' \gg 1$ into the ring as its new head); (vi) set $\mathtt{LINK}(j) \leftarrow p$, $W_{l'} \leftarrow j$ (thus inserting j into the watch list of l'); (vii) set $j \leftarrow j'$.

[The tricky part here is to remember that t can be zero in step (v).]

131. For example, the author tried selecting a variable x_k for which $s_{2k} \cdot s_{2k+1}$ is maximum, where s_l is the length of l's watch list plus ε, and the parameter ε was 0.1. This reduced the runtime for *waerden*(3, 10; 97) to 139.8 gigamems, with 8.6 meganodes. Less dramatic effects occurred with *langford*(13): 56.2 gigamems, with 10.8 meganodes, versus 99.0 gigamems if the *minimum* $s_{2k} \cdot s_{2k+1}$ was chosen instead.

132. The unsatisfiable clauses $(\bar{x}_1 \vee x_2)$, $(x_1 \vee \bar{x}_2)$, $(\bar{x}_3 \vee x_4)$, $(x_3 \vee \bar{x}_4)$, ..., $(\bar{x}_{2n-1} \vee x_{2n})$, $(x_{2n-1} \vee \bar{x}_{2n})$, $(\bar{x}_{2n-1} \vee \bar{x}_{2n})$, $(x_{2n-1} \vee x_{2n})$ cause it to investigate all 2^n settings of x_1, x_3, ..., x_{2n-1} before encountering a contradiction and repeatedly backtracking.

(Incidentally, the successive move codes make a pretty pattern. If the stated clauses are ordered randomly, the algorithm runs significantly faster, but it still apparently needs nonpolynomial time. What is the growth rate?)

133. (a) Optimum backtrack trees for n-variable SAT problems can be calculated with $\Theta(n3^n)$ time and $\Theta(3^n)$ space by considering all 3^n partial assignments, "bottom up." In this 9-variable problem we obtain a tree with 67 nodes (the minimum) if we branch first on x_3 and x_5, then on x_6 if $x_3 \neq x_5$; unit clauses arise at all other nodes.

(b) Similarly, the worst tree turns out to have 471 nodes. But if we require the algorithm to branch on a unit clause whenever possible, the worst size is 187. (Branch first on x_1, then x_4, then x_7; avoid opportunities for unit clauses.)

134. Let each `BIMP` list be accessed by `ADDR`, `BSIZE`, `CAP`, and `K` fields, where `ADDR` is the starting address in `MEM` of a block that's able to store `CAP` items, and $\mathtt{CAP} = 2^{\mathtt{K}}$; `ADDR` is a multiple of `CAP`, and `BSIZE` is the number of items currently in use. Initially $\mathtt{CAP} = 4$, $\mathtt{K} = 2$, $\mathtt{BSIZE} = 0$, and `ADDR` is a convenient multiple of 4. The $2n$ `BIMP` tables therefore occupy $8n$ slots initially. If `MEM` has room for 2^M items, those tables can be allocated so that the doubly linked lists $\mathtt{AVAIL}[k]$ initially contain $a_k = (0 \text{ or } 1)$ available blocks of size 2^k for each k, where $2^M - 8n = (a_{M-1} \ldots a_1 a_0)_2$.

Resizing is necessary when $\mathtt{BSIZE} = \mathtt{CAP}$ and we need to increase `BSIZE`. Set $a \leftarrow \mathtt{ADDR}$, $k \leftarrow \mathtt{K}$, $\mathtt{CAP} \leftarrow 2^{k+1}$, and let $b \leftarrow a \oplus 2^k$ be the address of a's buddy. If b is a free block of size 2^k, we're in luck: We remove b from $\mathtt{AVAIL}[k]$; then if $a\ \&\ 2^k = 0$, nothing needs to be done, otherwise we copy `BSIZE` items from a to b and set $\mathtt{ADDR} \leftarrow b$.

In the unlucky case when b is either reserved or free of size $< 2^k$, we set p to the address of the first block in $\mathtt{AVAIL}[k']$, where $\mathtt{AVAIL}[t]$ is empty for $k < t < k'$ (or we panic if `MEM`'s capacity is exceeded). After removing p from $\mathtt{AVAIL}[k']$, we split off new free blocks of sizes 2^{k+1}, ..., $2^{k'-1}$ if $k' > k+1$. Finally we copy `BSIZE` items from block a to block p, set $\mathtt{ADDR} \leftarrow p$, and put a into $\mathtt{AVAIL}[k]$. (We needn't try to "collapse" a with its buddy, since the buddy isn't free.)

135. They're the complements of the literals in $\mathtt{BIMP}(\bar{l})$.

empty list
Knuth
move codes
randomly
asymptotics
unit clause
Resizing

swap
virtual unswapping
BSTAMP
BST(l)
stamping

136. Before, $\{(1,2),(4,2),(4,5),(5,1),(5,7),(6,9)\}$; after, $\{(1,2),(4,2),(6,9)\}$.

137. If p in a TIMP list points to the pair (u,v), let's write $u = \mathtt{U}(p)$ and $v = \mathtt{V}(p)$.

(a) Set $N \leftarrow n - G$, $x \leftarrow \mathtt{VAR}[N]$, $j \leftarrow \mathtt{INX}[X]$, $\mathtt{VAR}[j] \leftarrow x$, $\mathtt{INX}[x] \leftarrow j$, $\mathtt{VAR}[N] \leftarrow X$, $\mathtt{INX}[X] \leftarrow N$. Then do the following for $l = 2X$ and $l = 2X+1$, and for all p in TIMP(l): $u \leftarrow \mathtt{U}(p)$, $v \leftarrow \mathtt{V}(p)$, $p' \leftarrow \mathtt{LINK}(p)$, $p'' \leftarrow \mathtt{LINK}(p')$; $s \leftarrow \mathtt{TSIZE}(\bar u)-1$, $\mathtt{TSIZE}(\bar u) \leftarrow s$, $t \leftarrow$ pair s of TIMP($\bar u$); if $p' \neq t$, swap pairs by setting $u' \leftarrow \mathtt{U}(t)$, $v' \leftarrow \mathtt{V}(t)$, $q \leftarrow \mathtt{LINK}(t)$, $q' \leftarrow \mathtt{LINK}(q)$, $\mathtt{LINK}(q') \leftarrow p'$, $\mathtt{LINK}(p) \leftarrow t$, $\mathtt{U}(p') \leftarrow u'$, $\mathtt{V}(p') \leftarrow v'$, $\mathtt{LINK}(p') \leftarrow q$, $\mathtt{U}(t) \leftarrow v$, $\mathtt{V}(t) \leftarrow \bar l$, $\mathtt{LINK}(t) \leftarrow p''$. Then set $s \leftarrow \mathtt{TSIZE}(\bar v)-1$, $\mathtt{TSIZE}(\bar v) \leftarrow s$, $t \leftarrow$ pair s of TIMP($\bar v$); if $p'' \neq t$, swap pairs by setting $u' \leftarrow \mathtt{U}(t)$, $v' \leftarrow \mathtt{V}(t)$, $q \leftarrow \mathtt{LINK}(t)$, $q' \leftarrow \mathtt{LINK}(q)$, $\mathtt{LINK}(q') \leftarrow p''$, $\mathtt{LINK}(p') \leftarrow t$, $\mathtt{U}(p'') \leftarrow u'$, $\mathtt{V}(p'') \leftarrow v'$, $\mathtt{LINK}(p'') \leftarrow q$, $\mathtt{U}(t) \leftarrow \bar l$, $\mathtt{V}(t) \leftarrow u$, $\mathtt{LINK}(t) \leftarrow p$.

Notice that we do *not* make the current pairs of TIMP(l) inactive. They won't be accessed by the algorithm until it needs to undo the swaps just made.

(b) In VAR and in each TIMP list, the active entries appear first. The inactive entries follow, in the same order as they were swapped out, because inactive entries never participate in swaps. Therefore we can reactivate the most-recently-swapped-out entry by simply increasing the count of active entries. We must, however, be careful to do this "virtual unswapping" in precisely the reverse order from which we did the swapping.

Thus, for $l = 2X+1$ and $l = 2X$, and for all p in TIMP(l), proceeding in the reverse order from (a), we set $u \leftarrow \mathtt{U}(p)$, $v \leftarrow \mathtt{V}(p)$, $\mathtt{TSIZE}(\bar v) \leftarrow \mathtt{TSIZE}(\bar v)+1$, and $\mathtt{TSIZE}(\bar u) \leftarrow \mathtt{TSIZE}(\bar u)+1$.

(The number N of free variables increases implicitly, because $N + E = n$ in step L12. Thus nothing needs to be done to VAR or INX.)

138. Because $\bar v \in \mathtt{BIMP}(\bar u)$, (62) will be used to make u nearly true. That loop will *also* make v nearly true, because $v \in \mathtt{BIMP}(u)$ is equivalent to $\bar u \in \mathtt{BIMP}(\bar v)$.

139. Introduce a new variable BSTAMP analogous to ISTAMP, and a new field BST(l) analogous to IST(l) in the data for each literal l. At the beginning of step L9, set $\mathtt{BSTAMP} \leftarrow \mathtt{BSTAMP}+1$, then set $\mathtt{BST}(l) \leftarrow \mathtt{BSTAMP}$ for $l = \bar u$ and all $l \in \mathtt{BIMP}(\bar u)$. Now, if $\mathtt{BST}(\bar v) \neq \mathtt{BSTAMP}$ and $\mathtt{BST}(v) \neq \mathtt{BSTAMP}$, do the following for all $w \in \mathtt{BIMP}(v)$: If w is fixed in context NT (it must be fixed true, since $\bar w$ implies $\bar v$), do nothing. Otherwise if $\mathtt{BST}(\bar w) = \mathtt{BSTAMP}$, perform (62) with $l \leftarrow u$ and exit the loop on w (because $\bar u$ implies both w and $\bar w$). Otherwise, if $\mathtt{BST}(w) \neq \mathtt{BSTAMP}$, append w to BIMP($\bar u$) and u to BIMP($\bar w$). (Of course (63) must be invoked when needed.)

Then increase BSTAMP again, and do the same thing with u and v reversed.

140. Unfortunately, no: We might have $\Omega(n)$ changes to BSIZE on each of $\Omega(n)$ levels of the search tree. However, the ISTACK will never have more entries than the total number of cells in all BIMP tables (namely 2^M in answer 134).

141. Suppose $\mathtt{ISTAMP} \leftarrow (\mathtt{ISTAMP}+1) \bmod 2^e$ in step L5. If $\mathtt{ISTAMP} = 0$ after that operation, we can safely set $\mathtt{ISTAMP} \leftarrow 1$ and $\mathtt{IST}(l) \leftarrow 0$ for $2 \le l \le 2n+1$. (A similar remark applies to BSTAMP and BST(l) in answer 139.)

142. (The following operations, performed after BRANCH[d] is set in step L2, will also output '|' to mark levels of the search where no decision was made.) Set $\mathtt{BACKL}[d] \leftarrow F$, $r \leftarrow k \leftarrow 0$, and do the following while $k < d$: While $r < \mathtt{BACKF}[k]$, output '$6+(R_r \& 1)$' and set $r \leftarrow r+1$. If $\mathtt{BRANCH}[k] < 0$, output '|'; otherwise output '$2\mathtt{BRANCH}[k]+(R_r \& 1)$' and set $r \leftarrow r+1$. While $r < \mathtt{BACKL}[k+1]$, output '$4+(R_r \& 1)$' and set $r \leftarrow r+1$. Then set $k \leftarrow k+1$.

Knuth
undoing

143. The following solution treats KINX and KSIZE as the unmodified algorithm treats TIMP and TSIZE. It deals in a somewhat more subtle way with CINX and CSIZE: If clause c originally had size k, and if j of its literals have become false while none have yet become true, CSIZE(c) will be $k - j$, but the nonfalse literals will not necessarily appear at the beginning of list CINX(c). As soon as j reaches $k-2$, or one of the literals becomes true, clause c becomes inactive and it disappears from the KINX tables of all free literals. The algorithm won't look at CINX(c) or CSIZE(c) again until it unfixes the literal that deactivated c. Thus a big clause is inactive if and only if it has been satisfied (contains a true literal) or has become binary (has at most two nonfalse literals).

We need to modify only the three steps that involve TIMP. The modified step L1, call it L1′, inputs the big clauses in a straightforward way.

Step L7′ removes the formerly free variable X from the data structures by first deactivating all of the active big clauses that contain L: For each of the KSIZE(L) numbers c in KINX(L), and for each of the CSIZE(c) free literals u in CINX(c), we swap c out of u's clause list as follows: Set $s \leftarrow$ KSIZE(u) $- 1$, KSIZE(u) $\leftarrow s$; find $t \le s$ with KINX(u)[t] $= c$; if $t \ne s$ set KINX(u)[t] $\leftarrow$ KINX(u)[s] and KINX(u)[s] $\leftarrow c$. [*Heuristic:* If the number of free literals remaining in c is small compared to c's original size, for example if say 15 or 20 original literals have become false, the remaining nonfalse literals can usefully be swapped into the first CSIZE(c) positions of CINX(c) when c is being deactivated. The author's experimental implementation does this when CSIZE(c) is at most θ times the original size, where the parameter θ is normally 25/64.]

Then step L7′ updates clauses for which L has become false: For each of the KSIZE($\bar{L}$) numbers c in KINX($\bar{L}$), set $s \leftarrow$ CSIZE(c) $- 1$ and CSIZE(c) $\leftarrow s$; if $s = 2$, find the two free literals (u, v) in CINX(c), swap them into the first positions of that list, put them on a temporary stack, and swap c out of the clause lists of u and v as above.

Finally, step L7′ does step L8′ = L8 for all (u, v) on the temporary stack. [The maximum size of that stack will be the maximum of KSIZE(l) over all l, after step L1′; so we allocate memory for that stack as part of step L1′.]

In step L12′ we set $L \leftarrow R_E$, $X \leftarrow |L|$, and reactivate the clauses that involve X as follows: For each of the KSIZE($\bar{L}$) numbers c in KINX($\bar{L}$), proceeding in reverse order from the order used in L7′, set $s \leftarrow$ CSIZE(c), CSIZE(c) $\leftarrow s + 1$; if $s = 2$, swap c back into the clause lists of v and u, where $u =$ CINX(c)[0] and $v =$ CINX(c)[1]. For each of the KSIZE(L) numbers c in KINX(L), and for each of the CSIZE(c) free literals u in CINX(c), again proceeding in reverse order from the order used in L7′, swap c back into the clause list of u. The latter operation simply increases KSIZE(u) by 1.

144. False; $h'(l) = 0.1$ if and only if the *complement*, $\bar{l}$, doesn't appear in any clause.

145. By symmetry we know that $h(l) = h(\bar{l}) = h(10 - l)$ for $1 \le l \le 9$ at depth 0, and the BIMP tables are empty. The first five rounds of refinement respectively give $(h(1), \ldots, h(5)) = (4.10, 4.10, 6.10, 6.10, 8.10)$, $(5.01, 4.59, 6.84, 6.84, 7.98)$, $(4.80, 4.58, 6.57, 6.57, 8.32)$, $(4.88, 4.54, 6.72, 6.67, 8.06)$, and $(4.85, 4.56, 6.63, 6.62, 8.23)$, slowly converging to the limiting values

$$(4.85810213, 4.55160111, 6.66761920, 6.63699698, 8.16778057).$$

When $d = 1$, however, the successively refined values of $(h(1), h(\bar{1}), \ldots, h(4), h(\bar{4}))$ are erratic and divergent: (2.10, 8.70, 3.10, 6.40, 3.10, 13.00, 3.10, 10.70), (5.53, 3.33, 9.05, 2.58, 5.40, 5.57, 8.24, 4.83), (1.43, 9.60, 2.32, 10.06, 1.30, 16.96, 1.97, 15.54), (8.04, 1.42, 12.31, 1.29, 7.45, 2.39, 11.91, 1.81), (0.32, 14.19, 0.42, 15.63, 0.30, 25.67, 0.43, 24.17).

They eventually oscillate between limits that favor either positive or negative literals:

$$(0.1012, 17.4178, 0.1019, 19.7351, 0.1015, 31.6345, 0.1021, 30.4902) \quad\text{and}\quad (10.3331, 0.1538, 15.8485, 0.1272, 9.6098, 0.1809, 15.4207, 0.1542).$$

[Equations (64) and (65), which were inspired by survey propagation, first appeared in unpublished work of S. Mijnders, B. de Wilde, and M. J. H. Heule in 2010. The calculations above indicate that we needn't take $h(l)$ too seriously, although it does seem to yield good results in practice. The author's implementation also sets $h'(l) \leftarrow \Theta$ if the right-hand side of (65) exceeds a threshold parameter Θ, which is 20.0 by default.]

146. Good results have been obtained with the simple formula $h(l) = \varepsilon + \mathtt{KSIZE}(\bar{l}) + \sum_{u\in\mathtt{BIMP}(l),\,u\text{ free}} \mathtt{KSIZE}(\bar{u})$, which estimates the potential number of big-clause reductions that occur when l becomes true. The parameter ε is typically set to 0.001.

147. ∞, 600, 60, 30, 30.

148. If a problem is easy, we don't care if we solve it in 2 seconds or in .000002 seconds. On the other hand if a problem is so difficult that it can be solved only by looking ahead more than we can accomplish in a reasonable time, we might as well face the fact that we won't solve it anyway. There's no point in looking ahead at 60 variables when $d = 60$, because we won't be able to deal with more than 2^{50} or so nodes in any reasonable search tree.

149. The idea is to maintain a binary string $\mathtt{SIG}(x)$ for each variable x, representing the highest node of the search tree in which x has participated. Let $b_j = [\mathtt{BRANCH}[j] = 1]$, and set $\sigma \leftarrow b_0 \ldots b_{d-1}$ at the beginning of step L2, $\sigma \leftarrow b_0 \ldots b_d$ at the beginning of step L4. Then x will be a participant in step X3 if and only if $\mathtt{SIG}(x)$ is a prefix of σ.

We update $\mathtt{SIG}(x)$ when $x = |u|$ or $x = |v|$ in step L9, by setting $\mathtt{SIG}(x) \leftarrow \sigma$ unless $\mathtt{SIG}(x)$ is a prefix of σ. The initial value of $\mathtt{SIG}(x)$ is chosen so that it is never a prefix of any possible σ.

(Notice that $\mathtt{SIG}(x)$ needn't change when backtracking. In practice we can safely maintain only the first 32 bits of σ and of each string $\mathtt{SIG}(x)$, together with their exact lengths, because lookahead computations need not be precise. In answer 143, updates occur not in step L9 but in step L7$'$; they are done for all literals $u \neq \overline{L}$ that appear in any big clause containing $\overline{L}$ that is being shortened for the first time.)

150. Asserting 7 at level 22 will also 22fix $\bar{1}$, because of the clause 147. Then $\bar{1}$ will 22fix 3 and 9, which will 22fix $\bar{2}$ and $\bar{6}$, then $\bar{8}$; and clause 258 becomes false. Therefore $\bar{7}$ becomes proto true; and (62) makes 3, 6, 9 all proto true, contradicting 369.

151. For example, one such arrangement is

$$\begin{array}{r|cccccccccccccccc} l: & 2 & \bar{8} & 9 & 3 & \bar{1} & 6 & \bar{7} & \bar{4} & 4 & 7 & \bar{6} & 1 & \bar{3} & \bar{9} & 8 & \bar{2} \\ o(l): & 4 & 2 & 10 & 14 & 6 & 16 & 8 & 12 & 22 & 26 & 18 & 28 & 20 & 24 & 32 & 30 \end{array}.$$

[Digraphs that are obtainable in this way are called "partial orderings of dimension ≤ 2," or *permutation posets*. We've actually seen them in exercise 5.1.1–11, where the set of arcs was represented as a set of *inversions*. Permutation posets have many nice properties, which we shall study in Section 7.4.2. For example, if we reverse the order of the list and complement the offsets, we reverse the directions on the arrows. All but two of the 238 connected partially ordered sets on six elements are permutation posets. Unfortunately, however, permutation posets don't work well with lookahead when they aren't also forests. For example, after 10fixing '9' and its consequences, we would want to remove those literals from the R stack when 14fixing '3'; see (71). But then we'd want them back when 6fixing '$\bar{1}$'.]

survey propagation
Mijnders
de Wilde
Heule
threshold parameter
partial orderings of dimension ≤ 2
permutation posets
inversions
partially ordered sets

152. A single clause such as '12' or '123' would be an example, except that the autarky test in step X9 would solve the problem before we ever get to step X3. The clauses $\{12\bar{3}, 1\bar{2}3, \bar{1}2\bar{3}, \bar{1}\bar{2}3, 245, 3\bar{4}\bar{5}\}$ do, however, work: Level 0 branches on x_1, and level 1 discovers an autarky with b and c both true but returns $l = 0$. Then level 2 finds all clauses satisfied, although both of the free variables x_4 and x_5 are newbies.

[Indeed, the absence of free participants means that the fixed-true literals form an autarky. If TSIZE(l) is nonzero for any free literal l, some clause is unsatisfied. Otherwise all clauses are satisfied unless some free l has an unfixed literal $l' \in$ BIMP(l).]

153. Make the CAND array into a heap, with an element x of *least* rating $r(x)$ at the top (see Section 5.2.3). Then, while $C > C_{\max}$, delete the top of the heap (namely CAND[0]).

154. The child $\longrightarrow$ parent relations in the subforest will be $d \longrightarrow c \longrightarrow a$, $b \longrightarrow a$, $\bar{c} \longrightarrow \bar{d}$, and either $\bar{a} \longrightarrow \bar{b}$ or $\bar{a} \longrightarrow \bar{c}$. Here's one suitable sequence, using the latter:

preorder	$\bar{b}$	a	b	c	d	$\bar{d}$	$\bar{c}$	$\bar{a}$
2·postorder	2	10	4	8	6	16	14	12

155. First construct the dependency graph on the $2C$ candidate literals, by extracting a subset of arcs from the BIMP tables. (This computation needn't be exact, because we're only calculating heuristics; an upper bound can be placed on the number of arcs considered, so that we don't spend too much time here. However, it is important to have the arc $u \longrightarrow v$ if and only if $\bar{v} \longrightarrow \bar{u}$ is also present.)

Then apply Tarjan's algorithm [see Section 7.4.1, or SGB pages 512–519]. If a strong component contains both l and $\bar{l}$ for some l, terminate with a contradiction. Otherwise, if a strong component contains more than one literal, choose a representative l with maximum $h(l)$; the other literals of that component regard l as their parent. Be careful to ensure that l is a representative if and only if $\bar{l}$ is also a representative.

The result will be a sequence of candidate literals $l_1 l_2 \ldots l_S$ in topological order, with $l_i \longrightarrow l_j$ only if $i > j$. Compute the "height" of each l_j, namely the length of the longest path from l_j to a sink. Then every literal of height $h > 0$ has a predecessor of height $h - 1$, and we let one such predecessor be its parent in the subforest. Every literal of height 0 (a sink) has a null parent. Traversal of this subforest in double order (exercise 2.3.1–18) now makes it easy to build the LL table in preorder while filling the LO table in postorder.

156. If $\bar{l}$ doesn't appear in any clause of F, then $A = \{l\}$ is clearly an autarky.

157. Well, *any* satisfying assignment is an autarky. But more to the point is the autarky $\{1, 2\}$ for $F = \{1\bar{2}3, \bar{1}24, \bar{3}\bar{4}\}$.

158. BIMP(l) and TIMP(l) will be empty, so w will be zero when Algorithm X looks ahead on l. Thus l will be forced true, at depth $d = 0$. (But pure literals that arise in subproblems for $d > 0$ won't be detected unless they're among the preselected candidates.)

159. (a) False (consider $A = \{1\}$, $F = \{1, 2, \bar{1}2\}$); but true if we assume that $F \mid A$ is computed as a multiset (so that $F \mid A$ would be $\{2, 2\} \not\subseteq F$ in that example).

(b) True: Suppose $A = A' \cup A''$, $A' \cap A'' = \emptyset$, and A'' or $\overline{A}''$ touches $C \in F \mid A'$. Then $C \cap A' = \emptyset$ and $C \cup C' \in F$, where $C' \subseteq \overline{A}'$. Since A or $\overline{A}$ touches $C \cup C'$, some $a \in C \cup C'$ is in A; hence $a \in A''$.

160. (a) If the gray clauses are satisfiable, let all black literals be true. [Notice, incidentally, that the suggested example coloring works like a charm in (7).]

autarky test
autarky
heap
heuristics
Tarjan
SGB
height
sink
sink: a vertex with no successor
double order
preorder
postorder
satisfying assignment
candidates
multiset

(b) Given any set A of strictly distinct literals, color l black if $l \in A$, white if $\bar{l} \in A$, otherwise gray. Then A is an autarky if and only if condition (a) holds.

[E. A. Hirsch, *Journal of Automated Reasoning* **24** (2000), 397–420.]

161. (a) If F' is satisfiable, so is F. If F is satisfiable with at least one blue literal false, so is F'. If F is satisfiable with all the blue literals true, make all the black literals true (but keep gray literals unchanged). Then F' is satisfied, because every clause of F' that contains a black or blue literal is true, hence every clause that contains a white literal is true; the remaining clauses, whose literals are only orange and gray, each contain at least one true gray literal. [The black-and-blue condition is equivalent to saying that A is a *conditional autarky*, namely an autarky of $F \mid L$. Tseytin's notion of "extended resolution" is a special case, because the literals of A and L need not appear in F. See S. Jeannicot, L. Oxusoff, and A. Rauzy, *Revue d'intelligence artificielle* **2** (1988), 41–60, Section 6; O. Kullmann, *Theoretical Comp. Sci.* **223** (1999), 1–72, Sections 3, 4, and 14.]

(b) Without affecting satisfiability, we are allowed to add or delete any clause $C = (a \vee \bar{l}_1 \vee \cdots \vee \bar{l}_q)$ for which all clauses containing $\bar{a}$ also contain l_1 or $\cdots$ or l_q. (Such a clause is said to be "blocked" with respect to a, because C produces nothing but tautologies when it is resolved with clauses that contain $\bar{a}$.)

(c) Without affecting satisfiability, we are allowed to add or delete any or all of the clauses $(\bar{l} \vee a_1)$, …, $(\bar{l} \vee a_p)$, if A is an autarky of $F \mid l$; that is, we can do this if A is *almost* an autarky, in the sense that every clause that touches $\overline{A}$ but not A contains l.

(d) Without affecting satisfiability, we are allowed to add or delete the clause $(\bar{l} \vee a)$ whenever every clause that contains $\bar{a}$ also contains l.

162. Construct a "blocking digraph" with $l' \hookrightarrow l$ when every clause that contains literal $\bar{l}$ also contains l'. (If l is a pure literal, we'll have $l' \hookrightarrow l$ for *all* l'; this case can be handled separately. Otherwise all in-degrees will be less than k in a kSAT problem, and the blocking digraph can be constructed in $O(k^2 m)$ steps if there are m clauses.)

(a) Then $(l \vee l')$ is a blocked binary clause if and only if $\bar{l} \hookrightarrow l'$ or $\bar{l}' \hookrightarrow l$. (Hence we're allowed in such cases to add both $\bar{l} \longrightarrow l'$ and $\bar{l}' \longrightarrow l$ to the *dependency* digraph.)

(b) Also $A = \{a, a'\}$ is an autarky if and only if $a \hookrightarrow a' \hookrightarrow a$. (Moreover, any strong component $\{a_1, \ldots, a_t\}$ with $t > 1$ is an autarky of size t.)

163. Consider the recurrence relations $T_n = 1 + \max(T_{n-1}, T_{n-2}, 2U_{n-1})$, $U_n = 1 + \max(T_{n-1}, T_{n-2}, U_{n-1} + V_{n-1})$, $V_n = 1 + U_{n-1}$ for $n > 0$, with $T_{-1} = T_0 = U_0 = V_0 = 0$. We can prove that T_n, U_n, V_n are upper bounds on the step counts, where U_n refers to cases where F is known to have a nonternary clause, and V_n refers to cases when $s = 1$ and R2 was entered from R3: The terms T_{n-1} and T_{n-2} represent autarky reductions in step R2; otherwise the recursive call in R3 costs U_{n-1}, not T_{n-1}, because at least one clause contains $\bar{l}_s$. We also have $V_n = 1 + U_{n-1}$, not $1 + T_{n-1}$, because the preceding step R3 either had a clause containing l_2 not l_1 or a clause containing $\bar{l}_1$ not $\bar{l}_2$.

Fibonacci numbers provide the solution: $T_n = 2F_{n+2} - 3 + [n = 0]$, $U_n = F_{n+3} - 2$, $V_n = F_{n+2} - 1$. [Algorithm R is a simplification of a procedure devised by B. Monien and E. Speckenmeyer, *Discrete Applied Mathematics* **10** (1985), 287–295, who introduced the term "autarky" in that paper. A Stanford student, Juan Bulnes, had discovered a Fibonacci-bounded algorithm for 3SAT already in 1976; his method was, however, unattractive, because it also required $\Omega(\phi^n)$ *space*.]

164. If $k < 3$, $T_n = n$ is an upper bound; so we may assume that $k \geq 3$. Let $U_n = 1 + \max(T_{n-1}, T_{n-2}, U_{n-1} + V_{n-1,1}, \ldots, U_{n-1} + V_{n-1,k-2})$, $V_{n,1} = 1 + U_{n-1}$, and $V_{n,s} = 1 + \max(U_{n-1}, T_{n-2}, U_{n-1} + V_{n-1,s-1})$ for $s > 1$, where $V_{n,s}$ refers to an entry at R2 from R3. The use of U_{n-1} in the formula for $V_{n,s}$ is justified, because the

Hirsch
conditional autarky
lookahead autarky clauses, see black and blue
Tseytin
extended resolution
Jeannicot
Oxusoff
Rauzy
Kullmann
blocked clauses
tautologies
resolved
blocking digraph
pure literal
dependency digraph
strong component
recurrence relations
Fibonacci numbers
Monien
Speckenmeyer
autarky
Bulnes

previous R3 either had a clause containing l_{s+1} not l_s or one containing $\bar{l}_s$ not $\bar{l}_{s+1}$. One can show by induction that $V_{n,s} = s + U_{n-1} + \cdots + U_{n-s}$, $U_n = V_{n,k-1}$; and $T_n = U_n + U_{n-k} + 1 = 2U_{n-1} + 1$ if $n \ge k$. For example, the running time when $k = 4$ is bounded by Tribonacci numbers, whose growth rate 1.83929^n comes from the root of $x^3 = x^2 + x + 1$.

165. Clause $\bar{1}\bar{3}\bar{4}$ in the example tells us that $1, 3, 4 \notin A$. Then $13\bar{6}$ implies $6 \notin A$. But $A = \{2, 5\}$ works, so it is maximum. There always is a maximum (not just maximal) positive autarky, because the union of positive autarkies is a positive autarky.

Each clause $(v_1 \vee \cdots \vee v_s \vee \bar{v}_{s+1} \vee \cdots \vee \bar{v}_{s+t})$ of F, where the v's are positive, tells us that $v_1 \notin A$ and $\cdots$ and $v_s \notin A$ implies $v_{s+j} \notin A$, for $1 \le j \le t$. Thus it essentially generates t Horn clauses, whose core is the set of all positive literals not in any positive autarky. A simple variant of Algorithm 7.1.1C will find this core in linear time; namely, we can modify steps C1 and C5 in order to get t Horn clauses from a single clause of F.

[By complementing a subset of variables, and prohibiting another subset, we can find the largest autarky A contained in any given set of strictly distinct literals. This exercise is due to unpublished work of O. Kullmann, V. W. Marek, and M. Truszczyński.]

166. Assume first that $\texttt{PARENT}(l_0) = \Lambda$, so that $H(l_0) = 0$ at the beginning of X9 (see X6). Since $l_0 = \texttt{LL}[j]$ is not fixed in context T, we have $R_F = l_0$ by (62). And $A = \{R_F, R_{F+1}, \ldots, R_{E-1}\}$ is an autarky, because no clause touched by A or $\overline{A}$ is entirely false or contains two unfixed literals. Thus we're allowed to force l_0 true (which is what "do step X12 with $l \leftarrow l_0$" means).

On the other hand if $w = 0$ and $\texttt{PARENT}(l_0) = p$, so that $H(l_0) = H(p) > 0$ in X6, the set $A = \{R_F, \ldots, R_{E-1}\}$ is an autarky with respect to the clauses of $F \,|\, p$. Hence the additional clause $(l_0 \vee \bar{p})$ doesn't make the clauses any less satisfiable, by the black and blue principle. (Notice that $(\bar{l}_0 \vee p)$ is already a known clause; so in this case l_0 is essentially being made equal to its parent.)

[The author's implementation therefore goes further and includes the step

$$\texttt{VAL}[|l_0|] \;\leftarrow\; \texttt{VAL}[|p|] \oplus \big((l_0 \oplus p) \,\&\, 1\big), \qquad (*)$$

which promotes the truth degree of l_0 to that of p. This step violates the invariant relation (71), but Algorithm X doesn't rely on (71).]

167. If a literal l is fixed in context T during the lookahead, it is implied by l_0. In step X11 we have a case where l is also implied by $\bar{l}_0$; hence we're allowed to force its truth, if l isn't already proto true. In step X6, $\bar{l}_0$ is implied by l_0, so l_0 must be false.

168. The following method works well in `march`: Terminate happily if $F = n$. (At this point in Algorithm L, F is the number of fixed variables, all of which are really true or really false.) Otherwise find $l \in \{\texttt{LL}[0], \ldots, \texttt{LL}[S-1]\}$ with $l \bmod 2 = 0$ and maximum $\big(H(l) + .1\big)\big(H(l{+}1) + .1\big)$. If l is fixed, set $l \leftarrow 0$. (In that case, Algorithm X found at least one forced literal, although U is now zero; we want to do another lookahead before branching again.) Otherwise, if $H(l) > H(l{+}1)$, set $l \leftarrow l{+}1$. (A subproblem that is less reduced will tend to be more satisfiable.)

169. When a and b are positive, the function $f(x) = e^{-ax} + e^{-bx} - 1$ is convex and decreasing, and it has the unique root $\ln \tau(a,b)$. Newton's method for solving this equation refines an approximation x by computing $x' = x + f(x)/(ae^{-ax} + be^{-bx})$. Notice that x is less than the root if and only if $f(x) > 0$; furthermore $f(x) > 0$ implies $f(x') > 0$, because $f(x') > f(x) + (x' - x)f'(x)$ when f is convex. In particular we have $f(1/(a+b)) > 0$, because $f(0) = 1$ and $0' = 1/(a+b)$, and we can proceed as follows:

K1. [Initialize.] Set $j \leftarrow k \leftarrow 1$, $x \leftarrow 1/(a_1 + b_1)$.

Tribonacci numbers
Horn clauses
core
Kullmann
Marek
Truszczyński
black and blue principle
author
truth degree
VAL
invariant relation
march
convex
Newton's method

K2. [Done?] (At this point (a_j, b_j) is the best of $(a_1, b_1), \ldots, (a_k, b_k)$, and $e^{-a_j x} + e^{-b_j x} \geq 1$.) Terminate if $k = s$. Otherwise set $k \leftarrow k+1$, $x' \leftarrow 1/(a_k + b_k)$.

K3. [Find α, β.] If $x' < x$, swap $j \leftrightarrow k$ and $x \leftrightarrow x'$. Then set $\alpha \leftarrow e^{-a_j x'}$ and $\beta \leftarrow e^{-b_j x'}$. Go to K2 if $\alpha + \beta \leq 1$.

K4. [Newtonize.] Set $x \leftarrow x' + (\alpha+\beta-1)/(a_j\alpha+b_j\beta)$, $\alpha' \leftarrow e^{-a_k x'}$, $\beta' \leftarrow e^{-b_k x'}$, $x' \leftarrow x' + (\alpha' + \beta' - 1)/(a_k\alpha' + b_k\beta')$, and return to K3. ▌

(The floating point calculations should satisfy $e^u \leq e^v$ and $u+w \leq v+w$ when $u < v$.)

170. If the problem is unsatisfiable, Tarjan's algorithm discovers l and $\bar{l}$ in the same strong component. If it's satisfiable, Algorithm X finds autarkies (because w is always zero), thus forcing the value of all literals at depth 0.

171. It prevents double-looking on the same literal twice at the same search tree node.

172. When Algorithm Y concludes normally, we'll have $T = $ `BASE`+`LO[`j`]`, even though `BASE` has changed. This relation is assumed to be invariant in Algorithm X.

173. The run reported in the text, using nonoptimized parameters (see exercise 513), did 29,194,670 double-looks (that is, executions of step Y2), and exited 23,245,231 times to X13 in step Y8 (thus successfully forcing l_0 false in about 80% of those cases). Disabling Algorithm Y (i) increased the running time from 0.68 teramems to 1.13 teramems, with 24.3 million nodes. Disabling wraparound (ii) increased the time to 0.85 teramems, with 13.3 million nodes. Setting $Y = 1$, which disabled wraparound only in Algorithm Y, yielded 0.72 teramems, 11.3 meganodes. (Incidentally, the loops of Algorithm X wrapped around 40% of the time in the regular run, with a mean of 0.62 and maximum of 12; those of Algorithm Y had 20% wraparound, with a mean of 0.25; the maximum $Y = 8$ was reached only 28 times.) Disabling the lookahead forest (iii) gave surprisingly good results: 0.70 teramems, 8.5 meganodes; there were fewer nodes [hence a more discriminating lookahead], but more time spent per node because of duplicated effort, although strong components were not computed. (Structured problems that have numerous binary clauses tend to generate more helpful forests than random 3SAT problems do.) Disabling compensation resolvents (iv) made very little difference: 0.70 teramems, 9.9 meganodes. But disabling windfalls (v) raised the cost to 0.89 teramems and 13.5 meganodes. And branching on a random $l \in$ `LL` (vi) made the running time soar to 40.20 teramems, with 594.7 meganodes. Finally, disabling Algorithm X altogether (vii) was a disaster, leading to an estimated run time of well over 10^{20} mems.

The weaker heuristics of exercise 175 yield 3.09 teramems and 35.9 meganodes.

174. Setting Y to a huge value such as `PT` will never get to step Y2. (But for (ii), (iii), …, (vii) one must change the programs, not the parameters as they stand.)

175. Precompute the weights, by setting $K_2 = 1$ and $K_s \leftarrow \gamma K_{s-1} + .01$, for s between 3 and the maximum clause size. (The extra .01 keeps this from being zero.) The third line of (72) must change to "take account of c for all c in `KINX(`$\overline{L}$`)`," where that means "set $s \leftarrow$ `CSIZE(`c`)` $- 1$; if $s \geq 2$, set `CSIZE(`c`)` $\leftarrow s$ and $w \leftarrow w + K_s$; otherwise if all literals of c are fixed false, set a flag; otherwise if some literal u of c isn't fixed (there will be just one), put it on a temporary stack." Before performing the last line of (72), go to `CONFLICT` if the flag is set; otherwise, for each unfixed u on the temporary stack, set $W_i \leftarrow u$ and $i \leftarrow i+1$ and perform (62) with $l \leftarrow u$; go to `CONFLICT` if some u on the temporary stack is fixed false. (A "windfall" in this more general setting is a clause for which all but one literal has been fixed false as a consequence of l_0 being fixed true.)

Of course those changes to `CSIZE` need to be undone; a simulated false literal that has been "virtually" removed from a clause must be virtually put back. Fortunately,

floating point
Tarjan
autarkies
invariant
windfall
undone

the invariant relation (71) makes this task fairly easy: We set $G \leftarrow F$ in step X5, and insert the following restoration loop at the very beginning of (72): "While $G > F$, set $u \leftarrow R_{G-1}$; stop if u is fixed in context T; otherwise set $G \leftarrow G - 1$, and increase CSIZE(c) by 1 for all $c \in$ KINX($\bar{u}$)." The restoration loop should also be performed, with $T \leftarrow$ NT, just before terminating Algorithm X in steps X7 or X13.

[The additional step (∗) in answer 166 can't be used, because (71) is now crucial.]

Algorithm Y should change in essentially the same way as Algorithm X.

[See O. Kullmann, Report CSR 23-2002 (Swansea: Univ. of Wales, 2002), §4.2.]

176. (a) a_j — a_{j+1}, a_j — b_j, a_j — b_{j+1}, b_j — c_j, b_j — d_j, c_j — d_j, c_j — e_j, d_j — f_j, e_j — d_{j+1}, e_j — f_{j+1}, f_j — c_{j+1}, f_j — e_{j+1}.

(b) Let $(t_j, u_j, v_j, w_j, a_j, b_j, c_j, d_j, e_j, f_j)$ have colors $(1,2,1,1,1,2,1,3,3,2)$ when j is even, $(2,1,2,2,3,2,3,1,1,2)$ when j is odd. The lower bounds are obvious.

(c) Vertices a_j, e_j, f_j can't all have the same color, because b_j, c_j, d_j have distinct colors. Let α_j denote the colors of $a_je_jf_j$. Then $\alpha_j = 112$ implies $\alpha_{j+1} = 332$ or 233; $\alpha_j = 121$ implies $\alpha_{j+1} = 233$ or 323; $\alpha_j = 211$ implies $\alpha_{j+1} = 323$ or 332; $\alpha_j = 123$ implies $\alpha_{j+1} = 213$ or 321. Since $\alpha_1 = \alpha_{q+1}$, the colors of α_1 must be distinct, and we can assume that $\alpha_1 = 123$. But then α_j will be an odd permutation whenever j is even.

[See Rufus Isaacs, *AMM* **82** (1975), 233–234. Unpublished notes of E. Grinberg show that he had independently investigated the graph J_5 in 1972.]

177. There are 20 independent subsets of $V_j = \{a_j, b_j, c_j, d_j, e_j, f_j\}$ when $q > 1$; eight of them contain none of $\{b_j, c_j, d_j\}$ while four contain b_j. Let A be a 20×20 transition matrix, which indicates when $R \cup C$ is independent for each independent subset $R \subseteq V_j$ and $C \subseteq V_{j+1}$. Then I_q is trace(A^q); and the first eight values are 8, 126, 1052, 11170, 112828, 1159416, 11869768, 121668290. The characteristic polynomial of A, $x^{12}(x^2 - 2x - 1)(x^2 + 2x - 1)(x^4 - 8x^3 - 25x^2 + 20x + 1)$, has nonzero roots $\pm 1 \pm \sqrt{2}$ and ≈ -2.91, -0.05, $+0.71$, $+10.25$; hence $I_q = \Theta(r^q)$, where $r \approx 10.24811166$ is the dominant root. *Note:* The number of *kernels* of $L(J_q)$ is respectively 2, 32, 140, 536, 2957, 14336, 70093, 348872, for $1 \le q \le 8$, and its growth rate is $\approx 4.93^q$.

178. With the first ordering, the top $18k$ levels of the search tree essentially represent all of the ways to 3-color the subgraph $\{a_j, b_j, c_j, d_j, e_j, f_j \mid 1 \le j \le k\}$; and there are $\Theta(2^k)$ ways to do that, by answer 176. But with the second ordering, the top $6kq$ levels essentially represent all of the independent sets of the graph; and there are $\Omega(10.2^k)$ of those, by answer 177.

Empirically, Algorithm B needs respectively 1.54 megamems, 1.57 gigamems, and 1.61 teramems to prove unsatisfiability when $q = 9$, 19, and 29, using the first ordering; but it needs 158 gigamems already for $q = 5$ with the second! Additional clauses, which require color classes to be kernels (see answer 14), reduce that time to 492 megamems.

Algorithm D does badly on this sequence of problems: When $q = 19$, it consumes 37.6 gigamems, even with the "good" ordering. And when $q = 29$, its cyclic method of working somehow transforms the good ordering into a bad ordering on many of the variables at depths 200 or more. It shows no sign of being anywhere near completion even after spending a *petamem* on that problem!

Algorithm L, which is insensitive to the ordering, needs 2.42 megamems, 2.01 gigamems, and 1.73 teramems when $q = 9$, 19, and 29. Thus it appears to take $\Theta(2^q)$ steps, and to be slightly slower than Algorithm B as q grows, although exercise 232 shows that a clairvoyant lookahead procedure could theoretically do much better.

Algorithm C triumphs here, as shown in Fig. 49.

Kullmann
odd permutation
Isaacs
Grinberg
transition matrix
trace of a matrix: The sum of its diagonal eleme
kernels
kernels
comparison of SAT solvers

179. This is a straightforward exact cover problem. If we classify the solutions according to how many asterisks occur in each coordinate, it turns out that exactly (10, 240, 180, 360, 720, 480, 1440, 270, 200, 480) of them are respectively of type (00088, 00268, 00448, 00466, 02248, 02266, 02446, 04444, 22228, 22246).

By complementation, we see that 4380 choices of 8 clauses are unsatisfiable; hence $q_8 = 1 - 4380/\binom{80}{8} = 1 - 4380/28987537150 \approx 0.9999998$.

180. With N variables y_j, one for each possible clause C_j, the function $f(y_1, \ldots, y_N) = [\bigwedge\{C_j \mid y_j = 1\}$ is satisfiable$]$ is $\bigvee_x f_x(y)$, where $f_x(y) = [x$ satisfies $\bigwedge\{C_j \mid y_j = 1\}]$ is simply $\bigwedge\{\bar{y}_j \mid x$ makes C_j false$\}$. For instance if $k = 2$ and $n = 3$, and if C_1, C_7, C_{11} are the clauses $(x_1 \vee x_2)$, $(x_1 \vee \bar{x}_3)$, $(x_2 \vee \bar{x}_3)$, then $f_{001}(y_1, \ldots, y_{12}) = \bar{y}_1 \wedge \bar{y}_7 \wedge \bar{y}_{11}$.

Each function f_x has a very simple BDD, but of course the OR of 2^n of them will not be simple. This problem is an excellent example where no natural ordering of the clause variables is evident, but the method of sifting is able to reduce the BDD size substantially. In fact, the clauses for $k = 3$ and $n = 4$ can be ordered cleverly so that the corresponding 32-variable BDD for satisfiability has only 1362 nodes! The author's best result for $k = 3$ and $n = 5$, however, was a BDD of size 2,155,458. The coefficients of its generating function (exercise 7.1.4–25) are the desired numbers Q_m.

The largest such count, $Q_{35} = 3{,}449{,}494{,}339{,}791{,}376{,}514{,}416$, is so enormous that we could not hope to enumerate the relevant sets of 35 clauses by backtracking.

181. The previous exercise essentially computed the generating function $\sum_m Q_m z^m$; now we want the *double* generating function $\sum_{l,m} T_{l,m} w^l z^m$, where $T_{l,m}$ is the number of ways to choose m different k-clauses in such a way that these clauses are satisfied by exactly l vectors $x_1 \ldots x_n$. To do this, instead of taking the OR of the simple functions f_x, we compute the BDD base that contains all of the symmetric Boolean functions $S_l(f_{0\ldots0}, \ldots, f_{1\ldots1})$ for $0 \le l \le 2^n$, as follows (see exercise 7.1.4–49): Consider the subscript x to be a binary integer, so that the functions are f_x for $0 \le x < 2^n$. Start with $S_l = 0$ for $-1 \le l \le 2^n$, except that $S_0 = 1$. Then do the following for $x = 0, \ldots, 2^n - 1$ (in that order): Set $S_l = f_x ? \, S_{l-1} \colon S_l$ for $l = x + 1, \ldots, 0$ (in that order).

After this computation, the generating function for S_l will be $\sum_m T_{l,m} z^m$. In the author's experiments, the sifting algorithm found an ordering of the 80 clauses for $k = 3$ and $n = 5$ so that only about 6 million nodes were needed when x had reached 24; afterwards, however, sifting took too long, so it was turned off. The final BDD base had approximately 87 million nodes, with many nodes shared between the individual functions S_l. The total running time was about 22 gigamems.

182. $T_0 = 32$ and $T_1 = 28$ and $T_m = 0$ for $71 \le m \le 80$. Otherwise $\min T_m < \max T_m$.

183. Let $t_m = \Pr(T_m = 1)$, and suppose that we obtain clauses one by one until reaching an unsatisfiable set. The fact that t_m gets reasonably large suggests that we probably have accumulated a *uniquely* satisfiable set just before stopping. (That probability is $2^{-k} N \sum_m t_m/(N-m)$, which turns out to be ≈ 0.8853 when $k = 3$ and $n = 5$.)

However, except for the fact that both Figs. 42 and 43 are bell-shaped curves with roughly the same tendency to be relatively large or small at particular values of m, there is apparently no strong mathematical connection. The probabilities in Fig. 43 sum to 1; but the sum of probabilities in Fig. 42 has no obvious significance.

When n is large, uniquely satisfiable sets are encountered only rarely. The final set before stopping a.s. has at most $f(n)$ solutions, for certain functions f; but how fast does the smallest such f grow? [See D. J. Aldous, *J. Theoretical Probability* **4** (1991), 197–211, for related ideas.]

exact cover
sifting
backtracking
generating function
BDD base
symmetric Boolean functions
mux
if-then-else, see mux
uniquely
Aldous

184. The probability $\hat{q}_m$ is $\widehat{Q}_m/N^m$, where $\widehat{Q}_m$ counts the choices $(C_1,\ldots,C_m)$ for which $C_1 \wedge \cdots \wedge C_m$ is satisfiable. The number of such choices that involve t distinct clauses is $t!\,{m \brace t}$ times Q_t, because ${m \brace t}$ enumerates set partitions; see Eq. 3.3.2–(5).

185. $\hat{q}_m = \sum_{t=0}^{N} {m \brace t} t!\, q_t \binom{N}{t}/N^m \ge q_m \sum_{t=0}^{N} {m \brace t} t! \binom{N}{t}/N^m = q_m$.

186. $\sum_m \sum_t {m \brace t} t!\, q_t \binom{N}{t} N^{-m}$ can be summed on m, since $\sum_m {m \brace t} N^{-m} = 1/(N-1)^{\underline{t}}$ by Eq. 1.2.9–(28). Similarly, the derivative of 1.2.9–(28) shows that $\sum_m m{m \brace t}N^{-m} = (N/(N-1)+\cdots+N/(N-t))/(N-1)^{\underline{t}}$.

187. In this special case, $q_m = [0 \le m < N]$ and $p_m = [m = N]$; hence $S_{n,n} = N = 2^n$ (and the variance is zero). By (78), we also have $\widehat{S}_{n,n} = NH_N$; indeed, the coupon collector's test (exercise 3.3.2–8) is an equivalent way to view this situation.

188. Now $q_m = 2^m n^{\underline{m}}/(2n)^{\underline{m}}$. It follows by (78) that $\widehat{S}_{1,n} = \sum_{m=0}^{n} 2^m n^{\underline{m}}/(2n-1)^{\underline{m}}$, because $N = 2n$. The identity $2^m n^{\underline{m}}/(2n-1)^{\underline{m}} = 2q_m - q_{m+1}$ yields the surprising fact that $\widehat{S}_{1,n} = (2q_0-q_1)+(2q_1-q_2)+\cdots = 1+S_{1,n}$; and we also have $\widehat{S}_{1,n}-1 = \frac{2n}{2n-1}S_{1,n-1}$. Hence, by induction, we obtain the (even more surprising) closed forms

$$S_{1,n} = 4^n \Big/ \binom{2n}{n}, \qquad \widehat{S}_{1,n} = 4^n \Big/ \binom{2n}{n} + 1.$$

So random 1SAT problems become unsatisfiable after $\sqrt{\pi n} + O(1)$ clauses, on average.

189. With the autosifting method in the author's experimental BDD implementation, the number of BDD nodes, given a sequence of m distinct clauses when $k = 3$ and $n = 50$, increased past 1000 when m increased from 1 to about 30, and it tended to peak at about 500,000 when m was slightly more than 100. Then the typical BDD size fell to about 50,000 when $m = 150$, and to only about 500 when $m = 200$.

BDD methods break down when n is too large, but when they apply we can count the total number of solutions remaining after m steps. In the author's tests with $k = 3$, $n = 50$, and $m = 200$, this number varied from about 25 to about 2000.

190. For example, $S_1(x_1,\ldots,x_n)$ can't be expressed in $(n-1)$CNF: All clauses of length $n-1$ that are implied by $S_1(x_1,\ldots,x_n)$ are also implied by $S_{\le 1}(x_1,\ldots,x_n)$.

191. Let $f(x_0,\ldots,x_{2^n-1}) = 1$ if and only if $x_0\ldots x_{2^n-1}$ is the truth table of a Boolean function of n variables that is expressible in kCNF. This function f is the conjunction of 2^n constraints $c(t)$, for $0 \le t = (t_0\ldots t_{2^n-1})_2 < 2^n$, where $c(t)$ is the following condition: If $x_t = 0$, then $\bigvee\{x_y \mid 0 \le y < 2^n,\ (y \oplus t)\,\&\,m = 0\}$ is 0 for some n-bit pattern m that has $\nu m = k$. By combining these constraints we can compute the BDD for f when $n = 4$ and $k = 3$; it has 880 nodes, and 43,146 solutions.

Similarly we have the following results, analogous to those in Section 7.1.1:

	$n=0$	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$
1CNF	2	4	10	28	82	244	730
2CNF	2	4	16	166	4,170	224,716	24,445,368
3CNF	2	4	16	256	43,146	120,510,132	4,977,694,100,656

And if we consider equivalence under complementation and permutation, the counts are:

1CNF	2	3	4	5	6	7	8
2CNF	2	3	6	14	45	196	1,360
3CNF	2	3	6	22	253	37,098	109,873,815

192. (a) $S(p) = \sum_{m=0}^{N} p^m(1-p)^{N-m}Q_m$. (b) We have $\int_0^N (t/N)^m(1-t/N)^{N-m}\,dt = N\mathrm{B}(m+1, N-m+1) = \frac{N}{N+1}\big/\binom{N}{m}$, by exercises 1.2.6–40 and 41; hence $\overline{S}_{k,n} = \frac{N}{N+1}\sum_{m=0}^{N} q_m = \frac{N}{N+1}S_{k,n}$. [See B. Bollobás, *Random Graphs* (1985), Theorem II.4.]

set partitions
Stirling subset numbers
coupon collector's test
autosifting
sifting
symmetric Boolean functions
symmetric threshold functions
truth table
Boolean functions in kCNF
BDD
Bollobás

194. A similar question, about proofs of *un*satisfiability when $\alpha > \limsup_{n\to\infty} S_{3,n}/n$, is also wide open.

195. $\mathrm{E}\,X = 2^n \Pr(0\ldots0 \text{ satisfies all}) = 2^n(1-2^{-k})^m = \exp\bigl(n\ln 2 + m\ln(1-2^{-k})\bigr) < 2\exp(-2^{-k-1}n\ln 2)$. Thus $S_k(\lfloor(2^k\ln 2)n\rfloor, n) = \Pr(X>0) \le \exp(-\Omega(n))$. [*Discrete Applied Math.* **5** (1983), 77–87. Conversely, in *J. Amer. Math. Soc.* **17** (2004), 947–973, D. Achlioptas and Y. Peres use the second moment principle to show that $\bigl(2^k\ln 2 - O(k)\bigr)n$ random kSAT clauses are almost always satisfiable by vectors x with $\nu x \approx n/2$. Careful study of "covering assignments" (see exercise 364) leads to the sharp bounds

$$2^k\ln 2 - \frac{1+\ln 2}{2} - O(2^{-\frac{k}{3}}) \le \liminf_{n\to\infty}\alpha_k(n) \le \limsup_{n\to\infty}\alpha_k(n) \le 2^k\ln 2 - \frac{1+\ln 2}{2} + O(2^{-\frac{k}{3}});$$

see A. Coja-Oghlan and K. Panagiotou, `arXiv:1310.2728` [math.CO] (2013), 48 pages.]

196. The probability is $\bigl((n-t)^{\underline{k}}/n^{\underline{k}}\bigr)^{\alpha n+O(1)} = e^{-kt\alpha}(1+O(1/n))$ that $\alpha n + O(1)$ random kSAT clauses omit t given letters. Let $p = 1-(1-e^{-k\alpha})^k$. By inclusion and exclusion, the first clause will be easy with probability $p(1+O(1/n))$, and the first two will both be easy with probability $p^2(1+O(1/n))$. Thus if $X = \sum_{j=1}^m [\text{clause } j \text{ is easy}]$, we have $\mathrm{E}\,X = pm + O(1)$ and $\mathrm{E}\,X^2 = p^2m^2 + O(m)$. Hence, by Chebyshev's inequality, $\Pr(|X-pm| \ge r\sqrt{m}) = O(1/r^2)$.

197. By Stirling's approximation, $\ln q(a,b,A,B,n) = nf(a,b,A,B) + g(a,b,A,B) - \frac12\ln 2\pi n - (\delta_{an} - \delta_{(a+b)n}) - (\delta_{bn} - \delta_{(b+B)n}) - (\delta_{An} - \delta_{(a+A)n}) - (\delta_{Bn} - \delta_{(A+B)n}) - \delta_{(a+b+A+B)n}$, where δ_n is positive and decreasing. And we must have $f(a,b,A,B) \le 0$, since $q(a,b,A,B,n) \le 1$. The O estimate is uniform when $0 < \delta \le a,b,A,B \le M$.

198. Consider one of the N^M possible sequences of M 3SAT clauses, where $N = 8\binom{n}{3}$ and $M = 5n$. By exercise 196 it contains $g = 5(1-(1-e^{-15})^3)n + O(n^{3/4})$ easy clauses, except with probability $O(n^{-1/2})$. Those clauses, though rare, don't affect the satisfiability; and all $\binom{M}{g}$ of the ways to insert them among the $r = M - g$ others are equally likely, so they tend to dampen the transition.

Let $l \le r$ be maximum so that the first l noneasy clauses are satisfiable, and let $p(l,r,g,m)$ be the probability that, when drawing m balls from an urn that contains g green balls and r red balls, at most l balls are red. Then $S_3(m,n) = \sum p(l,r,g,m)/N^M$ and $S_3(m',n) = \sum p(l,r,g,m')/N^M$, summed over all N^M sequences.

To complete the proof we shall show that

$$p(l,r,g,m+1) = p(l,r,g,m) - O(n^{-1/2}) \qquad \text{when } 3.5n < m < 4.5n;$$

hence $S_3(m+1,n) = S_3(m,n) - O(n^{-1/2})$, $S_3(m,n) - S_3(m',n) = O((m'-m)n^{-1/2})$. Notice that $p(l,r,g,m) = p(l,r,g,m+1)$ when $m < l$ or $m > l+g$; thus we may assume that l lies between $3.4n$ and $4.6n$. Furthermore the difference

$$d_m = p(l,r,g,m) - p(l,r,g,m+1) = \frac{\binom{m}{l}\binom{r+g-m-1}{r-l-1}}{\binom{r+g}{r}} = \frac{\binom{m}{l}\binom{r+g-m}{r-l}}{\binom{r+g}{r}}\frac{r-l}{r+g-m}$$

has a decreasing ratio $d_m/d_{m-1} = \bigl(m/(m-l)\bigr)\bigl((l+g+1-m)/(r+g-m)\bigr)$ when m increases from l to $l+g$. So $\max d_m$ occurs at $m \approx l(r+g)/r$, where this ratio is ≈ 1. Now exercise 197 applies with $a = l/n$, $b = \rho g/n$, $A = (r-l)/n$, $B = (1-\rho)g/n$, $\rho = l/r$.

[D. B. Wilson, in *Random Structures & Algorithms* **21** (2002), 182–195, showed that similar methods apply to many other threshold phenomena.]

Achlioptas
Peres
second moment principle
covering assignments
Coja-Oghlan
Panagiotou
inclusion and exclusion
Chebyshev's inequality
Stirling's approximation
balls and urns
urns and balls
Wilson
threshold phenomena

inclusion and exclusion
second moment principle

199. (a) Given the required letters $\{a_1,\ldots,a_t\}$, there are m ways to place the leftmost a_1, then $m-1$ ways to place the leftmost a_2, and so on; then there are at most N ways to fill in each of the remaining $m-t$ slots.

(b) By inclusion and exclusion: There are $(N-k)^m$ words that omit k of the letters.

(c) $N^{-m}\sum_k\binom{t}{k}(-1)^k\sum_j\binom{m}{j}N^{m-j}(-k)^j=\sum_j\binom{m}{j}(-1)^{j+t}N^{-j}A_j$, where $A_j=\sum_k\binom{t}{k}(-1)^{t-k}k^j=\left\{{j \atop t}\right\}t!$ by Eq. 1.2.6–(53).

200. (a) The unsatisfiable digraph must contain a strong component with a path

$$\bar l_t\longrightarrow l_1\longrightarrow\cdots\longrightarrow l_t\longrightarrow l_{t+1}\longrightarrow\cdots\longrightarrow l_l=\bar l_t,$$

where l_1, …, l_t are strictly distinct. This path yields an s-snare $(C;t,u)$ if we set s to the smallest index such that $|l_{s+1}|=|l_u|$ for some u with $1\le u<s$.

(b) No: $(x\vee y)\wedge(\bar y\vee x)\wedge(\bar x\vee y)$ and $(x\vee y)\wedge(\bar y\vee x)\wedge(\bar x\vee\bar y)$ are both satisfiable.

(c) Apply exercise 199(a) with $t=s+1$, $N=2n(n-1)$; note that $m^{\underline{s+1}}\le m^{s+1}$.

201. (a) Set $(l_i,l_{i+1})\leftarrow(x_1,x_2)$ or $(\bar x_2,\bar x_1)$, where $0\le i<2t$ (thus $4t$ ways).

(b) Set $(l_i,l_{i+1},l_{i+2})\leftarrow(x_1,x_2,x_3)$ or $(\bar x_3,\bar x_2,\bar x_1)$, where $0\le i<2t$; also $(\bar l_1,l_t,l_{t+1})$ or $(l_{t-1},l_t,\bar l_{2t-1})\leftarrow(x_1,x_2,x_3)$ or $(\bar x_3,\bar x_2,\bar x_1)$ (total $4t+4$ ways, if $t{>}2$).

(c) (l_1,l_{t-1},l_t) or $(\bar l_{2t-1},\bar l_{t+1},\bar l_t)\leftarrow(x_1,x_{t-1},x_t)$ or $(\bar x_{t-1},\bar x_1,x_t)$ (4 ways).

(d) l_i or $\bar l_{2t-i}\leftarrow x_i$ or $\bar x_{t-i}$, for $1\le i\le t$ (4 ways, if you understand this notation).

(e) By part (a), it is $2t\times 4t=8t^2$.

(f) Parts (b) and (c) combine to give $N(3,2)=(2t+2)\times(4t+4)+2\times 4=8(t^2+2t+2)$ when $t>2$. From part (d), $N(t,t)=8$. Also $N(2t-1,2t)=8$; this is the number of snakes that specify the same $2t$ clauses. (Incidentally, when $t=5$ the generating function $\sum_{q,r}N(q,r)\,w^q z^r$ is $1+200w^2z^1+(296w^3+7688w^4)z^2+(440w^4+12800w^5+55488w^6)z^3+(640w^5+12592w^6+66560w^7+31104w^8)z^4+(8w^5+736w^6+8960w^7+32064w^8+6528w^9)z^5+(32w^6+704w^7+4904w^8+4512w^9)z^6+(48w^7+704w^8+1232w^9)z^7+(64w^8+376w^9)z^8+80w^9z^9+8w^9z^{10}$.)

(g) The other l's can be set in at most $2^{2t-1-q}(n-q)^{\underline{2t-1-q}}=R/(2^q n^{\underline{q}})$ ways.

(h) We may assume that $r<2t$. The r chosen clauses divide into connected components, which are either paths or a "central" component that contains either $(\bar x_0\vee x_1)$ and $(\bar x_{t-1}\vee x_t)$ or $(\bar x_t\vee x_{t+1})$ and $(\bar x_{2t-1}\vee x_0)$. Thus q equals r plus the number of components, minus 1 if the central component includes a cycle. If the central component is present, we must set $l_t\leftarrow x_t$ or $\bar x_t$, and there are at most 8 ways to complete the mapping of that component. And $N(r,r)=16(r+1-t)$ for $t<r<2t$.

Cases with $k>0$ paths can be chosen in at most $\binom{2t+2}{2k}$ ways, because we choose the starting and ending points, and they can be mapped in at most $2^k k!\binom{2t+2}{2k}$ ways; so they contribute $\sum_{k>0}O(t^{4k}k/(k!^3n^k))=O(t^4/n)$ to $(2n)^r p_r$. The noncyclic central components, which can be chosen in $\Theta(t^4)$ ways, also contribute $O(t^4/n)$.

202. (a) $m(m-1)\ldots(m-r+1)/m^r\ge\bigl(1-\binom{r}{2}/m\bigr)$; $\bigl(2n(n-1)-r\bigr)^{m-r}/\bigl(2n(n-1)\bigr)^{m-r}\ge 1-(m-r)r/\bigl(2n(n-1)\bigr)$ when $r\le m<2n(n-1)$; and both factors are ≤ 1.

(b) The term of (95) for $r=0$ is 1 plus a negligible error. The contribution of $O(t^4/n)$ for $r>0$ is $O(n^{4/5+1/6-1})$, because $\sum_{r\ge 0}(1+n^{-1/6})^{-r}=n^{1/6}+1$. And the contributions of (96) to (95) for $r\ge t$ are exponentially small, because in that range we have $(1+n^{-1/6})^{-t}=\exp\bigl(-t\ln(1+n^{-1/6})\bigr)=\exp\bigl(-\Omega(n^{1/30})\bigr)$. Finally, then, by the second moment principle MPR–(22), $S_2(\lfloor n+n^{5/6}\rfloor,n)\le 1-\Pr(X>0)\le 1-(\mathrm{E}\,X)^2/(\mathrm{E}\,X^2)=1-1/\bigl((\mathrm{E}\,X^2)/(\mathrm{E}\,X)^2\bigr)=1-1/\bigl(1+O(n^{-1/30})\bigr)=O(n^{-1/30})$.

203. (a) $\mathrm{E}\,X=d^n\,\mathrm{E}\,X(1,\ldots,1)$, by symmetry; and $\mathrm{E}\,X(1,\ldots,1)=(1-p)^m$, because each set of q clauses is falsified with probability p. So $\mathrm{E}\,X=\exp\bigl((r\ln(1-p)+1)n\ln d\bigr)$ is exponentially small when $r\ln(1-p)+1<0$; and we know that $\Pr(X>0)\le\mathrm{E}\,X$.

(b) Let $\theta_s = \binom{s}{2}/\binom{n}{2} = \frac{s(s-1)}{n(n-1)}$, and consider a random constraint set, given that $X(1,\ldots,1) = 1$. With probability θ_s, both u and v have color 1 and the constraint is known to be satisfied. But with probability $1-\theta_s$, it holds with probability $\binom{d^2-2}{q}/\binom{d^2-1}{q}$. Thus $p_s = \bigl(\theta_s + (1-\theta_s)(d^2 - pd^2 - 1)/(d^2-1)\bigr)^m$. Tovey

(c) We have $\Pr(X>0) \ge d^n(1-p)^m/\,\mathrm{E}(X \mid X(1,\ldots,1)=1)$, from the inequality and symmetry; and the denominator is $\sum_{s=0}^{n}\binom{n}{s}(d-1)^{n-s}p_s$. We can replace p_s by the simpler value $p'_s = (1-p+ps^2/n^2)^m$, because $p_s < (\theta_s + (1-\theta_s)(1-p))^m = (1-p+\theta_s p)^m < p'_s$. And we can divide the simplified sum by $d^n(1-p)^m$.

(d) We have $\sum_{s=0}^{3n/d} t_s = e^{O(m/d^2)}\sum_{s=0}^{3n/d}\binom{n}{s}\bigl(\frac{1}{d}\bigr)^s\bigl(1-\frac{1}{d}\bigr)^{n-s}$, because $s^2/n^2 = O(1/d^2)$ when $s \le 3n/d$. This sum is $\ge 1-(e^2/27)^{n/d}$ by exercise 1.2.10–22; and the crucial assumption that $\alpha > \frac{1}{2}$ makes $m/d^2 \to 0$.

(e) Transition between increase and decrease occurs when $x_s \approx 1$; and we have

$$x_s = \frac{n-s}{s+1}\,\frac{1}{d-1}\left(1+\frac{(2s+1)p}{(1-p)n^2+ps^2}\right)^m \approx \exp\left(\ln\frac{1-\sigma}{\sigma} + \left(\frac{2pr\sigma}{1-p+p\sigma^2}-1\right)\ln d\right)$$

when $s = \sigma n$. Let $f(\sigma) = 2pr\sigma/(1-p+p\sigma^2) - 1$, and notice that $f'(\sigma) > 0$ for $0 \le \sigma < 1$ because $p \le \frac{1}{2}$. Furthermore our choice of r makes $f(\frac{1}{2}) < 0 < f(1)$. Setting $g(\sigma) = f(\sigma)/\ln\frac{\sigma}{1-\sigma}$, we seek values of σ with $g(\sigma) = 1/\ln d$. There are three such roots, because $g(1/N) \approx -f(0)/\ln N \ge 1/\ln N$; $g(\frac{1}{2} \pm 1/N) \approx \mp f(\frac{1}{2})N/4$; and $g(1-1/N) \approx f(1)/\ln N$.

(f) At the second peak, where $s = n - n/d^{f(1)}$, we have (see exercise 1.2.6–67)

$$t_s < \left(\frac{ned}{n-s}\right)^{n-s}\left(\frac{1}{d}\right)^n\left(1+\frac{p}{1-p}\right)^m = \exp\bigl((-\epsilon + O(1/d^{f(1)}))n\ln d\bigr),$$

which is exponentially small. And when $s = 3n/d$, $t_s < \bigl(\frac{ne}{sd}\bigr)^s e^{O(m/d^2)} = O((e/3)^{3n/d})$ is also exponentially small. Consequently $\sum_{s=3n/d}^{n} t_s$ is exponentially small.

[This derivation holds also when the random constraints are k-ary instead of binary, with $q = pd^k$ and $\alpha > 1/k$. See *J. Artificial Intelligence Res.* **12** (2000), 93–103.]

204. (a) If the original literals $\pm x_j$ that involve variable x_j correspond to $\sigma_1 X_{i(1)}, \ldots, \sigma_p X_{i(p)}$, with signs σ_h, add the clauses $(-\sigma_h X_{i(h)} \vee \sigma_{h^+} X_{i(h^+)})$ for $1 \le h \le p$ to enforce consistency, where $h^+ = 1+(h \bmod p)$. (This transformation, due to C. A. Tovey, works even in degenerate cases. For example, if $m = 1$ and if the given clause is $(x_1 \vee x_1 \vee \bar{x}_2)$, the transformed clauses are $(X_1 \vee X_2 \vee X_3)$, $(\bar{X}_1 \vee X_2)$, $(\bar{X}_2 \vee X_1)$, $(X_3 \vee \bar{X}_3)$.)

(b) After $F_0 = \{\epsilon\}$, $F_1 = F_0 \sqcup F_0$, $F_2 = F_0 \sqcup F_1$, $F_3 = F_0 \sqcup F_2$, $F_4 = F_3 \sqcup F'_3$, $F_5 = F_4 \sqcup F''_4$, always putting the new variable into the four shortest possible clauses, we get $F_5 = \{345,\ 2\bar{3}4,\ 1\bar{2}\bar{3},\ \bar{1}\bar{2}\bar{3},\ 3'\bar{4}5,\ 2'\bar{3}'\bar{4},\ 1'\bar{2}'\bar{3}',\ \bar{1}'\bar{2}'\bar{3}',\ 3''4''\bar{5},\ 2''\bar{3}''4'',\ 1''\bar{2}''\bar{3}'',\ \bar{1}''\bar{2}''\bar{3}'',\ 3'''\bar{4}''\bar{5},\ 2'''\bar{3}'''\bar{4}'',\ 1'''\bar{2}'''\bar{3}''',\ \bar{1}'''\bar{2}'''\bar{3}'''\}$.

(c) If we delete $\bar{1}\bar{2}\bar{3}$ from F_5 there are 288 solutions, namely $1 \wedge 2 \wedge 3 \wedge \bar{4} \wedge \bar{5} \wedge c' \wedge (4''?\ c'' \wedge \bar{3}''' \colon c''' \wedge \bar{3}'')$, where $c = \bar{2} \vee \bar{3}$.

(d) Add $\lceil m/2 \rceil$ disjoint clones of the 15 clauses of (c) to the $4m$ clauses of (a), giving $m + 15\lceil m/2\rceil$ 3-clauses and $3m$ 2-clauses that are satisfiable only if all literals cloned from $\bar{1}$, $\bar{2}$, or $\bar{3}$ are false. Each clone provides six such false literals $\{\bar{1}, \bar{1}, \bar{1}, \bar{2}, \bar{2}, \bar{3}\}$ without using any variable five times. So we can stick those literals into the 2-clauses, obtaining $\approx 11.5m$ 3-clauses in $N \approx 10.5m$ variables. (The new clauses have $288^{\lceil m/2\rceil}$ times as many solutions as the original ones. Can the ratio $N/m \approx 10.5$ be lowered?)

205. Let $F_0 = \{\epsilon\}$, $F_1 = F_0 \sqcup F_0$, $F_2 = F_0 \sqcup F_1$, $F_3 = F_0 \sqcup F_2$, $F_4 = F_0 \sqcup F_3$, $F_5 = F_1 \sqcup F_4$, $F_6 = F_0 \sqcup F_5$, $F_7 = F_0 \sqcup F_6$, $F_8 = F_4 \sqcup F'_7$, $F_9 = F_0 \sqcup F_8$, $F_{10} = F_7 \sqcup F'_9$,

$F_{11} = F_7 \sqcup F'_{10}$, $F_{12} = F_0 \sqcup F_{11}$, $F_{13} = F_9 \sqcup F''_{12}$, $F_{14} = F_{10} \sqcup F^{(3)}_{12}$, $F_{15} = F_{12} \sqcup F^{(4)}_{14}$, $F_{16} = F_{13} \sqcup F^{(6)}_{14}$, $F_{17} = F_{14} \sqcup F^{(7)}_{15}$, $F_{18} = F_{16} \sqcup F^{(13)}_{17}$. (Here '$x^{(3)}$' stands for '$x'''$', etc.) Then F_{18} consists of 257 unsatisfiable 4-clauses in 234 variables.

(Is there a shorter solution? This problem was first solved by J. Stříbrná in her M.S. thesis (Prague: Charles University, 1994), with 449 clauses. The $\sqcup$ method was introduced by S. Hoory and S. Szeider, *Theoretical Computer Science* **337** (2005), 347–359, who presented an unsatisfiable 5SAT problem that uses each variable at most 7 times. It's not known whether 7 can be decreased to 6 when every clause has size 5.)

206. Suppose F and F' are minimally unsatisfiable, and delete a clause of $F \sqcup F'$ that arose from F'; then we can satisfy $F \sqcup F'$ with x true.

Conversely, if $F \sqcup F'$ is minimally unsatisfiable, F and F' can't both be satisfiable. Suppose F is unsatisfiable but F' is satisfied by L'. Removing a clause of $F \sqcup F'$ that arose from F' is satisfiable only with x true; but then we can use L' to satisfy $F \sqcup F'$. Hence F and F' are both unsatisfiable. Finally, if $F \setminus C$ is unsatisfiable, so is $(F \sqcup F') \setminus (C \,|\, \bar{x})$, because any solution would satisfy either $F \setminus C$ or F'.

207. The five clauses of $C(x,y,z;a,b,c) = \{x\bar{a}b, y\bar{b}c, z\bar{c}a, abc, \bar{a}\bar{b}\bar{c}\}$ resolve to the single clause xyz. Thus $C(x,y,y;1,2,3) \cup C(x,\bar{y},\bar{y};4,5,6) \cup C(\bar{x},z,z;7,8,9) \cup C(\bar{x},\bar{z},\bar{z};a,b,c)$ is a solution. [K. Iwama and K. Takaki, *DIMACS* **35** (1997), 315–333, noted that the 16 clauses $\{\bar{x}\bar{y}\bar{z}\} \cup C(x,x,x;1,2,3) \cup C(y,y,y;4,5,6) \cup C(z,z,z;7,8,9)$ involve each variable exactly four times, and proved that no set of twelve clauses does so.]

208. Make m clones of all but one of the 20 clauses in answer 207, and put the other $3m$ cloned literals into the $3m$ binary clauses of answer 204(a). This gives $23m$ 3-clauses in which every literal occurs twice, except that the $3m$ literals $\bar{X}_i$ occur only once.

To complete the solution, we "pad" them with additional clauses that are always satisfiable. For example, we could introduce $3m$ more variables u_i, with new clauses $\bar{X}_i u_i \bar{u}_{i+1}$ for $1 \le i \le 3m$ and $\{u'_{3j}u'_{3j+1}u'_{3j+2}, \bar{u}'_{3j}\bar{u}'_{3j+1}\bar{u}'_{3j+2}\}$ for $1 \le j \le m$ (treating subscripts mod $3m$), where u'_i denotes (i even? u_i: $\bar{u}_i$).

209. Since the multiset of kt literals in any t clauses contains at least t different variables, the "marriage theorem" (Theorem 7.5.1M) implies that we can choose a different variable in each clause, easily satisfying it. [*Discr. Applied Math.* **8** (1984), 85–89.]

210. [P. Berman, M. Karpinski, A. D. Scott, *Electronic Colloquium on Computational Complexity* (2003), TR22.] This answer uses the magic number $\varepsilon = \delta^7 \approx 1/58$, where δ is the smallest root of $\delta((1-\delta^7)^6 + (1-\delta^7)^7) = 1$. We will assign random values to each variable so that Pr[all clauses are satisfied] > 0.

Let $\eta_j = (1-\varepsilon)^j/((1-\varepsilon)^j + (1-\varepsilon)^{13-j})$, and observe that $\eta_j \le \delta(1-\varepsilon)^j$ for $0 \le j \le 13$. If variable x occurs d^+ times and $\bar{x}$ occurs d^- times, let x be true with probability η_{d^-}, false with probability $1-\eta_{d^-} = \eta_{13-d^-} \le \delta(1-\varepsilon)^{13-d^-} \le \delta(1-\varepsilon)^{d^+}$.

Let $\mathrm{bad}(C)$ = [clause C is falsified by the random assignment], and construct the lopsidependency graph for these events as in exercise 351. Then, if the literals of $C = (l_1 \vee \cdots \vee l_7)$ have contrary appearances in $d_1, \ldots, d_7$ other clauses, we have

$$\Pr(\mathrm{bad}(C)) \le (\delta(1-\varepsilon)^{d_1}) \ldots (\delta(1-\varepsilon)^{d_7}) = \varepsilon(1-\varepsilon)^{d_1+\cdots+d_7} \le \varepsilon(1-\varepsilon)^{\mathrm{degree}(C)},$$

because C has at most $d_1 + \cdots + d_7$ neighbors. Theorem L, with parameter $\theta_i = \varepsilon$ for each event $\mathrm{bad}(C)$, now tells us that Pr[all m clauses are satisfied] $\ge (1-\varepsilon)^m$.

[See H. Gebauer, T. Szabó, and G. Tardos, *SODA* **22** (2011), 664–674, for asymptotic results that apply to kSAT as $k \to \infty$.]

Stříbrná
Hoory
Szeider
5SAT
resolution
Iwama
Takaki
marriage theorem
Berman
Karpinski
Scott
lopsidependency graph
Gebauer
Szabó
Tardos

4-cycles
exact cover
3D matching
sudoku
headers
Poisson probability
Franco
Jacquet
Knessl
Szpankowski
Goldberg

211. If m clauses in n variables are given, so that $3m = 4n$, let $N = 8n$. Consider N "colors" named jk or $\overline{jk}$, where $1 \le j \le n$ and k is one of the four clauses that contains $\pm x_j$. Let σ be a permutation on the colors, consisting of 4-cycles that involve the same variable, with the properties that (i) $(jk)\sigma = jk'$ for some k' and (ii) $(\overline{jk})\sigma = \overline{(jk)\sigma}$.

There are $4n$ vertices of K_N named jk, having the respective color lists

$$L(jk,1) = \{jk, \overline{jk}\}, \quad L(jk,2) = \{jk, (jk)\sigma\}, \quad L(jk,3) = \{\overline{jk}, (jk)\sigma\}.$$

The other $3m$ vertices of K_N are named a_k, b_k, c_k for each clause k. If that clause is, say, $x_2 \vee \bar{x}_5 \vee x_6$, the color lists are

$$L(a_k,1) = \{2k, \overline{5k}, 6k\}, \quad L(b_k,1) = L(c_k,1) = \{2k, \overline{2k}, 5k, \overline{5k}, 6k, \overline{6k}\};$$
$$L(a_k,2) = \{\overline{(2k)\sigma}\}, \quad L(b_k,2) = \{\overline{(5k)\sigma}\}, \quad L(c_k,2) = \{\overline{(6k)\sigma}\};$$
$$L(a_k,3) = \{\overline{(2k)\sigma^2}, (2k)\sigma\}, \quad L(b_k,3) = \{\overline{(5k)\sigma^2}, (5k)\sigma\}, \quad L(c_k,3) = \{\overline{(6k)\sigma^2}, (6k)\sigma\}.$$

Then $K_N \,\square\, K_3$ is list-colorable if and only if the clauses are satisfiable. (For example, $(jk,1)$ is colored $jk \iff ((jk)\sigma, 1)$ is colored $(jk)\sigma \iff (a_k, 1)$ is not colored jk.)

212. (a) Let $x_{ijk} = 1$ if and only if $X_{ij} = k$. [*Note:* Another equivalent problem is to find an exact cover of the rows $\{\,\{\mathrm{P}ij, \mathrm{R}ik, \mathrm{C}jk\} \mid p_{ij} = r_{ik} = c_{jk} = 1\}$. This is a special case of 3D matching; see the discussion of sudoku in Section 7.2.2.1. Incidentally, the 3D matching problem can be formulated as the problem of finding a binary tensor (x_{ijk}) such that $x_{ijk} \le y_{ijk}$ and $x_{i**} = x_{*j*} = x_{**k} = 1$, given (y_{ijk}).]

(b) $c_{31} = c_{32} = r_{13} = r_{14} = 0$ forces $x_{13*} = 0 \ne p_{13}$ when $r = c = \begin{pmatrix}1100\\0110\\0011\\1001\end{pmatrix}$, $p = \begin{pmatrix}1010\\1100\\0101\\0011\end{pmatrix}$.

(c) Make $L(I,J) = \{1, \ldots, N\}$ for $M < I \le N$, $1 \le J \le N$. It is well known (Theorem 7.5.1L) that a latin rectangle can always be extended to a latin square.

(d) Index everything by the set $\{1, \ldots, N\} \cup \bigcup_{I,J} \{(I,J,K) \mid K \in L(I,J)\}$. The elements (I,J,K) where $K = \min L(I,J)$ are called *headers*. Set $p_{ij} = 1$ if and only if (i) $i = j = (I,J,K)$ is not a header; or (ii) $i = (I,J,K)$ is a header, and $j = J$ or $j = (I,J,K')$ is not a header; or (iii) $j = (I,J,K)$ is a header, and $i = I$ or $i = (I,J,K')$ is not a header. Set $r_{ik} = c_{ik} = 1$ if and only if (i) $1 \le i,k \le N$; or (ii) $i = (I,J,K)$ and $k = (I,J,K')$, and if i is not a header then ($K' = K$ or K' is the largest element $< K$ in $L(I,J)$). [*Reference: SICOMP* **23** (1994), 170–184.]

213. The hinted probability is $\bigl(1-(1-p)^{n'}(1-q)^{n-n'}\bigr)^m$, where $n' = b_1 + \cdots + b_n$. Thus if $p \le q$, *every* x has probability at least $(1-(1-p)^n)^m$ of satisfying every clause. This is huge, unless n is small or m is large: If m is less than α^n, where α is any constant less than $1/(1-p)$, then when $n > -1/\lg(1-p)$ the probability $(1-(1-p)^n)^m > \exp(\alpha^n \ln(1-(1-p)^n)) > \exp(-2(\alpha(1-p))^n) > 1 - 2(\alpha(1-p))^n$ is exponentially close to 1. Nobody needs a SAT solver for such an easy problem.

Even if, say, $p = q = k/(2n)$, so that the average clause size is k, a clause is empty — hence unsatisfiable — with probability $e^{-k} + O(n^{-1})$; and indeed a clause has exactly r elements with the Poisson probability $e^{-k}k^r/r! + O(n^{-1})$ for fixed r. So the model isn't very relevant. [See J. Franco, *Information Proc. Letters* **23** (1986), 103–106.]

214. (a) $T(z) = ze^z + 2T(pz)(e^{(1-p)z} - 1)$.

(b) If $f(z) = \prod_{m=1}^{\infty}(1 - e^{(p-1)z/p^m})$ and $\tau(z) = f(z)T(z)e^{-z}$, we have $\tau(z) = zf(z) + 2\tau(pz) = zf(z) + 2pzf(pz) + 4p^2zf(p^2z) + \cdots$.

(c) See P. Jacquet, C. Knessl, and W. Szpankowski, *Combinatorics, Probability, and Computing* **23** (2014), 829–841. [The sequence $\langle T_n \rangle$ was first studied by A. T. Goldberg, *Courant Computer Science Report* **16** (1979), 48–49.]

215. Since any given $x_1 \ldots x_l$ is a partial solution in $\bigl(8\binom{n}{3} - \binom{l}{3}\bigr)^m$ of the $\bigl(8\binom{n}{3}\bigr)^m$ possible cases, level l contains $P_l = 2^l(1 - \frac{1}{8}l^3/n^3)^m$ nodes on the average. When $m = 4n$ and $n = 50$, the largest levels are $(P_{31}, P_{32}, \ldots, P_{36}) \approx (6.4, 6.9, 7.2, 7.2, 6.8, 6.2) \times 10^6$, and the mean total tree size $P_0 + \cdots + P_{50}$ is about 85.6 million.

If $l = 2tn$ and $m = \alpha n$ we have $P_l = 2^{f(t)n}$, where $f(t) = 2t + \alpha \lg(1-t^3) + O(1/n)$ for $0 \le t \le 1/2$. The maximum $f(t)$ occurs when $\ln 4 = 3\alpha t^2/(1-t^3)$, at which point $t = t_\alpha = \beta - \frac{1}{2}\beta^4 + \frac{5}{8}\beta^7 + O(\beta^{10})$, where $\beta = \sqrt{\ln 4/(3\alpha)}$; for example, $t_4 \approx 0.334$. Now

$$P_{L+k} = 2^E,\ \ E = (2t_\alpha + \alpha\lg(1-t_\alpha^3))n + 1 - 2t_\alpha - \frac{\gamma k^2}{n} + O\Bigl(\frac{1}{n}\Bigr) + O\Bigl(\frac{k^3}{n^2}\Bigr),\ \ \gamma = \frac{\alpha + t_\alpha \ln 2}{2\alpha t_\alpha},$$

when $L = 2t_\alpha n$; hence the expected total tree size is $\sqrt{\pi n/\gamma}\,P_L\bigl(1 + O(1/\sqrt{n})\bigr)$.

[This question was first studied by C. A. Brown and P. W. Purdom, Jr., *SICOMP* **10** (1981), 583–593; K. M. Bugrara and C. A. Brown, *Inf. Sciences* **40** (1986), 21–37.]

216. If the search tree has q two-way branches, it has fewer than $2nq$ nodes; we shall find an upper bound on $\mathrm{E}\,q$. Consider such branches after values have been assigned to the first l variables $x_1, \ldots, x_l$, and also to s additional variables $y_1, \ldots, y_s$ because of unit-clause forcing; the branch therefore occurs on level $t = l + s$. The values can be assigned in 2^t ways, and the y's can be chosen in $\binom{n-1-l}{s}$ ways. For $1 \le i \le s$ the m given clauses must contain $j_i \ge 1$ clauses chosen (with replacement) from the $F = \binom{t-1}{2}$ that force the value of y_i from other known values. The other $m - j_1 - \cdots - j_s$ must be chosen from the $R = 8\binom{n}{3} - sF - \binom{t}{3} - 2\binom{t}{2}(n-t)$ remaining clauses that aren't entirely false and don't force anything further. Thus the expected number of two-way branches is at most

$$P_{lt} = 2^t\binom{n-l-1}{s}\sum_{j_1,\ldots,j_s\ge 1}\binom{m}{j_1,\ldots,j_s,m-j}\frac{F^j R^{m-j}}{N^m},\ j = j_1 + \cdots + j_s,\ N = 8\binom{n}{3},$$

summed over $0 \le l \le t < n$. Let $b = F/N$ and $c = R/N$; the sum on $j_1, \ldots, j_s$ is

$$m!\,[z^m]\,(e^{bz}-1)^s e^{cz} = \sum_r \binom{s}{r}(-1)^{s-r}(c+rb)^m = s!\,c^m\sum_q\binom{m}{q}\left\{{q \atop s}\right\}\Bigl(\frac{b}{c}\Bigr)^q.$$

These values P_{lt} are almost all quite small when $m = 200$ and $n = 50$, rising above 100 only when $l \ge 45$ and $t = 49$; $\sum P_{lt} \approx 4404.7$.

If $l = xn$ and $t = yn$, we have $b \approx \frac{3}{8}y^2/n$ and $c \approx 1 - \frac{1}{8}(3(y-x)y^2 + y^3 + 6y^2(1-y))$. The asymptotic value of $[z^{\alpha n}]\,(e^{\beta z/n}-1)^{\delta n}e^{\gamma z}$ can be found by the saddle point method: Let ζ satisfy $\beta\delta e^\zeta/(e^\zeta - 1) + \gamma = \alpha\beta/\zeta$, and let $\rho^2 = \alpha/\zeta^2 - \delta e^\zeta/(e^\zeta-1)^2$. Then the answer is approximately $(e^\zeta-1)^{\delta n}e^{\gamma\zeta n/\beta}\sqrt{n}/\bigl(\sqrt{2\pi}\rho\beta(\zeta n/\beta)^{\alpha n+1}\bigr)$.

[For exact formulas and lower bounds, see *SICOMP* **12** (1983), 717–733. The total time to find all solutions grows approximately as $\bigl(2(\frac{7}{8})^\alpha\bigr)^n$ when $\alpha < 4.5$, according to H.-M. Méjean, H. Morel, and G. Reynaud, *SICOMP* **24** (1995), 621–649.]

217. True, unless both l and $\bar l$ belong to A or to B (making A or B tautological). For if L is a set of strictly distinct literals that covers both A and B, we know that neither A nor B nor L contains both l and $\bar l$; hence $L \setminus \{l, \bar l\}$ covers $(A\setminus\{l,\bar l\}) \cup (B \setminus \{l, \bar l\}) = C$.

(This generalization of resolution is, however, useless if $C \supseteq A$ or $C \supseteq B$, because a large clause is easier to cover than any of its subsets. Thus we generally assume that $l \in A$ and $\bar l \in B$, and that C isn't tautological, as in the text.)

218. x? B: A. [Hence $(x \vee A) \wedge (\bar x \vee B)$ always implies $A \vee B$.]

Brown
Purdom
Bugrara
Stirling subset numbers
asymptotic
saddle point method
Méjean
Morel
Reynaud
tautological
generalization of resolution

tautological
commutative
associative
McColl
refutation chain
pure literal
unnecessary branch
Knuth
Robinson
awkward

219. If C' or C'' is tautological ($\wp$), we define $\wp \diamond C = C \diamond \wp = C$. Otherwise, if there's a unique literal l such that C' has the form $l \vee A'$ and C'' has the form $\bar{l} \vee A''$, we define $C' \diamond C'' = A' \vee A''$ as in the text. If there are two or more such literals, strictly distinct, we define $C' \diamond C'' = \wp$. And if there are no such literals, we define $C' \diamond C'' = C' \vee C''$.

[This operation is obviously commutative but not associative. For example, we have $(\bar{x} \diamond \bar{y}) \diamond (x \vee y) = \wp$ while $\bar{x} \diamond (\bar{y} \diamond (x \vee y)) = \epsilon$.]

220. (a) True: If $C \subseteq C'$ and $C' \subseteq C''$ and $C'' \neq \wp$ then $C' \neq \wp$; hence every literal of C appears in C' and in C''. [The notion of subsumption goes back to a paper by Hugh McColl, *Proc. London Math. Soc.* **10** (1878), 16–28.]

(b) True: Otherwise we'd necessarily have $(C \diamond C') \vee \alpha \vee \alpha' \neq \wp$ and $C \neq \wp$ and $C' \neq \wp$ and $C \diamond C' \neq C \vee C'$; hence there's a literal l with $C = l \vee A$, $C' = \bar{l} \vee A'$, and the literals of $A \vee A' \vee \alpha \vee \alpha'$ are strictly distinct. So the result is easily checked, whether or not α or α' contains l or $\bar{l}$. (Notice that we always have $C \diamond C' \subseteq C \vee C'$.)

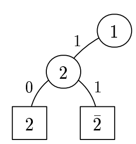
(c) False: $\bar{x}y \subseteq \wp$ but $x \diamond \bar{x}y = y \not\subseteq x = x \diamond \wp$. Also $\epsilon \subseteq \bar{x}$ but $x \diamond \epsilon = x \not\subseteq \epsilon = x \diamond \bar{x}$.

(d) Such examples are possible if $C \neq \epsilon$: We have $x, \bar{x} \vdash y$ (and also $x, \bar{x} \vdash \wp$), although the only clauses obtainable from x and $\bar{x}$ by resolution are x, $\bar{x}$, and ϵ. (On the other hand we do have $F \vdash \epsilon$ *if and only if* there's a refutation chain (104) for F.)

(e) Given a resolution chain $C'_1, \ldots, C'_{m+r}$, we can construct another chain $C_1, \ldots, C_{m+r}$ in which $C_i \subseteq C'_i$ for $1 \le i \le m+r$. Indeed, if $i > m$ and $C'_i = C'_j \diamond C'_k$, it's easy to see that either $C_j \diamond C_j$ or $C_k \diamond C_k$ or $C_j \diamond C_k$ will subsume C'_i.

(f) It suffices by (e) to prove this when $\alpha_1 = \cdots = \alpha_m = \alpha$; and by induction we may assume that $\alpha = l$ is a single literal. Given a resolution chain $C_1, \ldots, C_{m+r}$ we can construct another one $C'_1, \ldots, C'_{m+r}$ such that $C'_i = C_i \vee l$ for $1 \le i \le m$ and $C'_i \subseteq C_i \vee l$ for $m+1 \le i \le m+r$, with $C'_i = C'_j$ or C'_k or $C'_j \diamond C'_k$ whenever $C_i = C_j \diamond C_k$.

221. Algorithm A recognizes '1' as a pure literal, but then finds a contradiction because the *other* two clauses are unsatisfiable. The resolution refutation uses only the other two clauses. (This is an example of an unnecessary branch. Indeed, a pure literal never appears in a refutation tree, because it can't be canceled; see the next exercise.)



222. If A is an autarky that satisfies C, it also satisfies every clause on the path to ϵ from a source vertex labeled C, because all of the satisfied literals cannot simultaneously vanish. For the converse, see *Discrete Appl. Math.* **107** (2000), 99–137, Theorem 3.16.

223. (The author has convinced himself of this statement, but he has not been able to construct a formal proof.)

224. At every leaf labeled by an axiom A of $F \mid \bar{x}$ that is not an axiom of F, change the label to $A \cup x$; also include x in the labels of all this leaf's ancestors. We obtain a resolution tree in which the leaves are labeled by axioms of F. The root is labeled x, if any labels have changed; otherwise it is still labeled ϵ.

[See J. A. Robinson, *Machine Intelligence* **3** (1968), 77–94.]

225. Let's say that a regular resolution tree for clause A is *awkward* if at least one of its nodes resolves on one of the variables in A. An awkward tree T for A can always be transformed into a regular non-awkward tree T' for some clause $A' \subseteq A$, where T' is smaller than T. *Proof:* Suppose T is awkward, but none of its subtrees are. Without loss of generality we can find a sequence of subtrees $T_0, \ldots, T_p, T'_1, \ldots, T'_p$, where $T_0 = T$ and T_{j-1} for $1 \le j \le p$ is obtained from T_j and T'_j by resolving on the variable x_j; furthermore $x_p \in A$. We can assume that the labels of T_j and T'_j are A_j and A'_j, where $A_j = x_j \cup R_j$ and $A'_j = \bar{x}_j \cup R'_j$; hence $A_{j-1} = R_j \cup R'_j$. Let $B_p = A_p$; and for

$j = p-1, p-2, \ldots, 1$, let $B_j = B_{j+1}$ if $x_j \notin B_{j+1}$, otherwise obtain B_j by resolving B_{j+1} with A'_j. It follows by induction that $B_j \subseteq x_p \cup A_{j-1}$. Thus $B_1 \subseteq x_p \cup A_0 = A$, and we've derived B_1 with a non-awkward tree smaller than T.

Now we can prove more than was asked: If T is any resolution tree that derives clause A, and if $A \cup B$ is any clause that contains A, there's a non-awkward regular resolution tree T_r no larger than T that derives some clause $C \subseteq A \cup B$. The proof is by induction on the size of T: Suppose $A = A' \cup A''$ is obtained at the root of T by resolving the clauses $x \cup A'$ with $\bar{x} \cup A''$ that label the subtrees T' and T''. Find non-awkward regular trees T'_r and T''_r that derive C' and C'', where $C' \subseteq x \cup A' \cup B$ and $C'' \subseteq \bar{x} \cup A'' \cup B$. If $x \in C'$ and $\bar{x} \in C''$, we obtain the desired T_r by resolving T'_r and T''_r on x. Otherwise we can either let $C = C'$ and $T_r = T'_r$, or $C = C''$ and $T_r = T''_r$. [It's interesting to apply this construction to the highly irregular resolutions in (105).]

Kullmann
tautological
transitive law
hyperresolution

226. Initially α is the root, $C(\alpha) = \epsilon$, $\|\alpha\| = N$, and $s = 0$. If α isn't a leaf, we have $C(\alpha) = C(\alpha') \diamond C(\alpha'')$ where $x \in C(\alpha')$ and $\bar{x} \in C(\alpha'')$ for some variable x. The Prover names x, and changes $\alpha \leftarrow \alpha'$ or $\alpha \leftarrow \alpha''$ if the Delayer sets $x \leftarrow 0$ or $x \leftarrow 1$, respectively. Otherwise $\min(\|\alpha'\|, \|\alpha''\|) \le \|\alpha\|/2$, and the Prover can keep going.

227. The proof is by induction on the number of variables, n: If F contains the empty clause, the game is over, the Delayer has scored 0, and the root is labeled 0. Otherwise the Prover names x, and the Delayer considers the smallest possible labels (m, m') on the roots of refutations for $F \mid x$ and $F \mid \bar{x}$. If $m > m'$, the reply $x \leftarrow 0$ guarantees m points; and the reply $x \leftarrow *$ is no better, because $m' + 1 \le m$. If $m < m'$, the reply $x \leftarrow 1$ guarantees m'; and if $m = m'$, the reply $x \leftarrow *$ guarantees $m + 1$. Thus an optimum Delayer can always score at least as many points as the root label of any branch of a refutation tree constructed by the Prover. Conversely, if the Prover always names an optimal x, the Delayer can't do better.

(This exercise was suggested by O. Kullmann. One can compute the optimum score "bottom up" by considering all 3^n possible partial assignments as in answer 133.)

228. We need only assume the transitivity clauses T_{ijk} of (100) when $i < j$ and $k < j$. [Notice further that T_{ijk} is tautological when $i = j$ or $k = j$, thus useless for resolution.]

229. Using the binary-relation interpretation, these clauses say that $j \not\prec j$, that the transitive law "$i \prec j$ and $j \prec k$ implies $i \prec k$" holds whenever $i \le k$ and $j < k$, and that every j has a successor such that $j \prec k$. The latter axiom combines with the finiteness of m to imply that there must be a cycle $j_0 \prec j_1 \prec \cdots \prec j_{p-1} \prec j_p = j_0$.

Consider the *shortest* such cycle, and renumber the subscripts so that $j_p = \max\{j_0, \ldots, j_p\}$. We cannot have $p \ge 2$, because (100′) implies $j_{p-2} \prec j_p$, yielding a shorter cycle. Hence $p = 1$; but that contradicts (99).

230. Call the axioms I_j, T_{ijk}, and M_{jm} as in the text. If I_{j_0} is omitted, let $x_{ij} = [j = j_0]$ for all i and j. If $T_{i_0 j_0 k_0}$ is omitted, let $x_{ij} = [j \in A]$ for all $i \notin A = \{i_0, j_0, k_0\}$; also let $x_{i_0 j} = [j = j_0]$, $x_{j_0 j} = [j = k_0]$, and (if $i_0 \ne k_0$) $x_{k_0 j} = [j = i_0]$. Finally, if $M_{j_0 m}$ is omitted, let $x_{ij} = [p_i < p_j]$, where $p_1 \ldots p_m = 1 \ldots (j_0-1)(j_0+1) \ldots m j_0$. (The same construction shows that the clauses of answer 228 are minimally unsatisfiable.)

231. Since $G_{11} = M_{1m}$, we can assume that $j > 1$. Then $G_{(j-1)j} = G_{(j-1)(j-1)} \diamond I_{j-1}$. And if $1 \le i < j-1$ we have $G_{ij} = (\cdots((G_{(j-1)j} \diamond A_{ijj}) \diamond A_{ij(j+1)}) \diamond \cdots) \diamond A_{ijm}$, where $A_{ijk} = G_{i(j-1)} \diamond T_{i(j-1)k} = G_{ij} \vee \bar{x}_{(j-1)k}$. These clauses make it possible to derive $B_{ij} = (\cdots((G_{ij} \diamond T_{jij}) \diamond T_{ji(j+1)}) \diamond \cdots) \diamond T_{jim} = G_{jj} \vee \bar{x}_{ji}$ for $1 \le i < j$, from which we obtain $G_{jj} = (\cdots((M_{jm} \diamond B_{1j}) \diamond B_{2j}) \diamond \cdots) \diamond B_{(j-1)j}$. Finally $G_{mm} \diamond I_{mm} = \epsilon$.

232. It suffices to exhibit a backtrack tree of depth $6 \lg q + O(1)$. By branching on at most 6 variables we can find the color-triplet α_1 in answer 176(c). Cook Cook

Suppose we know that $\alpha_j = \alpha$ and $\alpha_{j+p} = \alpha'$, where α' cannot be obtained from α in p steps; this is initially true with $j = 1$, $\alpha = \alpha' = \alpha_1$, and $p = q$. If $p = 1$, a few more branches will find a contradiction. Otherwise at most 6 branches will determine α_l, where $l = j + \lfloor p/2 \rfloor$; and either α_l will be unreachable from α in $\lfloor p/2 \rfloor$ steps, or α' will be unreachable from α_l in $\lceil p/2 \rceil$ steps, or both. Recurse.

233. $C_9 = C_4 \diamond C_8$, $C_{10} = C_1 \diamond C_9$, $C_{11} = C_5 \diamond C_{10}$, $C_{12} = C_6 \diamond C_{10}$, $C_{13} = C_7 \diamond C_{11}$, $C_{14} = C_3 \diamond C_{12}$, $C_{15} = C_{13} \diamond C_{14}$, $C_{16} = C_2 \diamond C_{15}$, $C_{17} = C_4 \diamond C_{15}$, $C_{18} = C_8 \diamond C_{15}$, $C_{19} = C_{12} \diamond C_{17}$, $C_{20} = C_{11} \diamond C_{18}$, $C_{21} = C_{16} \diamond C_{19}$, $C_{22} = C_{20} \diamond C_{21}$.

234. Reply $x_{jk} \leftarrow *$ to any query that doesn't allow the Prover to violate (107). Then the Prover can violate (106) only after every hole has been queried.

235. Let $C(k, A) = (\bigvee_{j=0}^{k} \bigvee_{a \in A} x_{ja})$, so that $C(0, \{1, \ldots, m\}) = (x_{01} \vee \cdots \vee x_{0m})$ and $C(m, \emptyset) = \epsilon$. The chain consists of k stages for $k = 1, \ldots, m$, where stage k begins by deriving the clauses $\bar{x}_{ka} \vee C(k-1, A)$ from the clauses of stage $k-1$, for all $(m-k)$-element subsets A of $\{1, \ldots, m\} \setminus a$; every such clause requires k resolutions with (107). Stage k concludes by deriving $C(k, A)$ for all $(m-k)$-element subsets A of $\{1, \ldots, m\}$, each using one resolution from (106) and $k-1$ resolutions from the beginning of the stage. (See (103).) Thus stage k involves a total of $\binom{m}{m-k}(k^2 + k)$ resolutions.

For example, the resolutions when $m = 3$ successively yield $\overline{11}\,02\,03$, $\overline{12}\,01\,03$, $\overline{13}\,01\,02$; $01\,02\,11\,12$, $01\,03\,11\,13$, $02\,03\,12\,13$ (stage 1); $\overline{21}\,02\,11\,12$, $\overline{21}\,02\,12$, $\overline{21}\,03\,11\,13$, $\overline{21}\,03\,13$, $\overline{22}\,01\,12\,11$, $\overline{22}\,01\,11$, $\overline{22}\,03\,12\,13$, $\overline{22}\,03\,13$, $\overline{23}\,01\,13\,11$, $\overline{23}\,01\,11$, $\overline{23}\,02\,13\,12$, $\overline{23}\,02\,12$; $01\,11\,21\,22$, $01\,11\,21$, $02\,12\,22\,23$, $02\,12\,22$, $03\,13\,23\,22$, $03\,13\,23$ (stage 2); and $\overline{31}\,11\,21$, $\overline{31}\,21$, $\overline{31}$, $\overline{32}\,12\,22$, $\overline{32}\,22$, $\overline{32}$, $\overline{33}\,13\,23$, $\overline{33}\,23$, $\overline{33}$; $32\,33$, 33, ϵ (stage 3).

[Stephen A. Cook constructed such chains in 1972 (unpublished).]

236. The symmetry of the axioms should allow exhaustive verification by computer for $m = 2$, possibly also for $m = 3$. The construction certainly seems hard to beat. Cook conjectured in 1972 that any minimum-length resolution proof would include, for every subset S of $\{1, \ldots, m\}$, at least one clause C such that $\bigcup_{\pm x_{jk} \in C} \{k\} = S$.

237. The idea is to define $y_{jk} = x_{jk} \vee (x_{jm} \wedge x_{mk})$ for $0 \le j < m$ and $1 \le k < m$, thus reducing from m pigeons to $m-1$. First we append $6(m-1)(m-2)$ new clauses

$$(x_{jm} \vee z_{jk}) \wedge (x_{mk} \vee z_{jk}) \wedge (\bar{x}_{jm} \vee \bar{x}_{mk} \vee \bar{z}_{jk}) \wedge (\bar{x}_{jk} \vee y_{jk}) \wedge (y_{jk} \vee z_{jk}) \wedge (x_{jk} \vee \bar{y}_{jk} \vee \bar{z}_{jk}),$$

involving $2(m-1)(m-2)$ new variables y_{jk} and z_{jk}. Call these clauses $A_{jk}, \ldots, F_{jk}$.

Now if P_j stands for (106) and H_{ijk} for (107), we want to use resolution to derive $P'_j = (y_{j1} \vee \cdots \vee y_{j(m-1)})$ and $H'_{ijk} = (\bar{y}_{ik} \vee \bar{y}_{jk})$. First, P_j can be resolved with D_{j1}, $\ldots$, $D_{j(m-1)}$ to get $P'_j \vee x_{jm}$. Next, $P_m \diamond H_{jmm} = x_{m1} \vee \cdots \vee x_{m(m-1)} \vee \bar{x}_{jm}$ can be resolved with $G_{jk} = C_{jk} \diamond E_{jk} = \bar{x}_{jm} \vee \bar{x}_{mk} \vee y_{jk}$ for $1 \le k < m$ to get $P'_j \vee \bar{x}_{jm}$. One more step yields P'_j. (The intuitive "meaning" guides these maneuvers.)

From $B_{jk} \diamond F_{jk} = x_{jk} \vee x_{mk} \vee \bar{y}_{jk}$, we obtain $Q_{ijk} = \bar{x}_{ik} \vee \bar{y}_{jk}$ after resolving with H_{ijk} and H_{imk}. Then $(Q_{ijk} \diamond F_{ik}) \diamond A_{ik} = x_{im} \vee \bar{y}_{ik} \vee \bar{y}_{jk} = R_{ijk}$, say. Finally, $(R_{jik} \diamond H_{ijm}) \diamond R_{ijk} = H'_{ijk}$ as desired. (When forming R_{jik} we need Q_{jik} with $j > i$.)

We've done $5m^3 - 6m^2 + 3m$ resolutions to reduce m to $m-1$. Repeating until $m = 0$, with fresh y and z variables each time, yields ϵ after about $\frac{5}{4}m^4$ steps.

[See Stephen A. Cook, *SIGACT News* **8**, 4 (October 1976), 28–32.]

238. The function $(1 - cx)^{-x} = \exp(cx^2 + c^2x^3/2 + \cdots)$ is increasing and $> e^{cx^2}$. Setting $c = \frac{1}{2n}$, $W = \sqrt{2n \ln r}$, and $b = \lceil W \rceil$ makes $f \le r < \rho^{-b}$. Also $W \ge w(\alpha_0)$

when $n \ge w(\alpha_0)^2$ and $r \ge 2$; hence $w(\alpha_0 \vdash \epsilon) \le W + b \le \sqrt{8n \ln r} + 1$ as desired. The '-2' in the lemma handles the trivial cases that arise when $r < 2$.

(It is important to realize that we don't change n or W in the induction proof. Incidentally, the exact minimum of $W + b$, subject to $r = (1 - W/(2n))^{-b}$, occurs when

$$W = 2n(1 - e^{-2T(z)}) = 4nz + \frac{2nz^3}{3} + \cdots, \qquad b = \frac{\ln r}{2T(z)} = (\ln r)\Big(\frac{1}{2z} - \frac{1}{2} - \frac{z}{4} - \cdots\Big),$$

where $z^2 = (\ln r)/(8n)$ and $T(z)$ is the tree function. Thus it appears likely that the proof of Lemma B supports the stronger result $w(\alpha_0 \vdash \epsilon) < \sqrt{8n \ln r} - \frac{1}{2}\ln r + 1$.)

asymptotic methods
generating functions
tree function
complete binary tree
matching
boundary

239. Let α_0 consist of all 2^n nontautological clauses of length n. The shortest refutation is the complete binary tree with these leaves, because every nontautological clause must appear. Algorithm A shows that $2^n - 1$ resolutions suffice to refute any clauses in n variables; hence $\|\alpha_0 \vdash \epsilon\| = 2^n - 1$, and this is the worst case.

240. If A' has t elements and $\partial A'$ has fewer than t, the sequence of $5t$ integers f_{ij} for its neighbors must include at least $2t$ repeats of values seen earlier. (In fact there are at least $2t+1$ repeats, because $2t$ would leave at least t in the boundary; but the calculations are simpler with $2t$, and we need only a rather crude bound.)

The probability p_t that some such A' exists is therefore less than $\binom{m+1}{t}\binom{5t}{2t}\big(\frac{3t}{m}\big)^{2t}$, because there are $\binom{m+1}{t}$ ways to select A', $\binom{5t}{2t}$ to select the repeating slots, and at most $(3t)^{2t}$ out of m^{2t} ways to fill those slots. Also $\binom{m+1}{t} = \binom{m}{t} + \binom{m}{t-1} < 2\binom{m}{t}$ when $t \le \frac{1}{2}m$.

By exercise 1.2.6–67 we have $p_t \le 2\big(\frac{me}{t}\big)^t\big(\frac{5te}{2t}\big)^{2t}\big(\frac{3t}{m}\big)^{2t} = 2(ct/m)^t$, where $c = 225e^3/4 \approx 1130$. Also $p_0 = p_1 = 0$. Thus the sum of p_t for $t \le m/3000$ is less than $2\sum_{t=2}^{\infty}(c/3000)^t \approx .455$; and the probability of strong expansion exceeds .544.

241. If $0 < |A'| \le m/3000$, we can put one of its elements into a hole $b_k \in \partial A'$. Then we can place the other elements in the same way, since b_k isn't their neighbor.

242. The proof of Theorem B remains valid when these new axioms are added.

243. (a) The probability that F' has t elements and $V(F')$ has fewer than t is at most $\binom{\alpha n}{t}\binom{n}{t}\big(\frac{t}{n}\big)^{3t} \le \big(\frac{\alpha e^2 t}{n}\big)^t$. The sum of this quantity for $1 \le t \le \lg n$ is $O(n^{-1})$, and so is the sum for $\lg n \le t \le n/(2\alpha e^2)$.

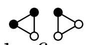
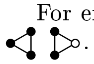
(b) If the condition in (a) holds, there's a matching from F' into $V(F')$, by Theorem 7.5.1M; hence we can satisfy F' by assigning to its variables, one by one. If F is unsatisfiable we'll therefore need to invoke more than $n/(2\alpha e^2)$ of its axioms.

(c) The probability p_t that F' has t elements and $2|V(F')| - 3|F'| < \frac{1}{2}|F'|$ is at most $\binom{\alpha n}{t}\binom{n}{\lambda t}\big(\frac{\lambda t}{n}\big)^{3t} \le \big(\alpha e^{1+\lambda}\lambda^{3-\lambda}(t/n)^{1/4}\big)^t$, where $\lambda = \frac{7}{4}$. We have $(e^{1+\lambda}\lambda^{3-\lambda})^4 < 10^6$; so $p_t < c^t$ when $t \le n'$, where $c < 1$, and $\sum_{t=n'/2}^{n'} p_t$ is exponentially small.

(d) Since $n' < n/(2\alpha e^2)$, every refutation a.s. contains a clause C with $n'/2 \le \mu(C) < n'$. The minimal axioms F' on which C depends have $|F'| = \mu(C)$. Let k be the number of "boundary" variables that occur in just one axiom of F'. If v is such a variable, we can falsify C and the axiom containing v, while the other axioms of F' are true; hence V must contain v or $\bar{v}$. We have $|V(F')| = k + |\text{nonboundary}| \le k + \frac{1}{2}(3|F'| - k)$, because each nonboundary variable occurs at least twice. Therefore $k \ge 2|V(F')| - 3|F'| \ge n'/4$, q.s. (Notice the similarities to the proof of Theorem B.)

244. We have $[A \cup B]^0 = [A]^0[B]^0 \cup [A]^1[B]^1$ and $[A \cup B]^1 = [A]^0[B]^1 \cup [A]^1[B]^0$, where concatenation of sets has the obvious meaning. These relations hold also when $A = \emptyset$ or $B = \emptyset$, because $[\emptyset]^0 = \{\epsilon\}$ and $[\emptyset]^1 = \emptyset$.

245. (a) When conditioning on e_{uv}, simply delete the edge u — v from G. When conditioning on $\bar{e}_{uv}$, also complement $l(u)$ and $l(v)$. The graph might become disconnected; in that case, there will be exactly two components, one even and one odd, with respect to the sums of their labels. The axioms for the even component are satisfiable and may be discarded.

For example, $\alpha(G)\,|\,\{\bar{b}, e\}$ corresponds to  while $\alpha(G)\,|\,\{\bar{b}, \bar{e}\}$ corresponds to . We toss out the left component in the first case, the right one in the other.

(b) If $C \in \alpha(v)$ we may take $V' = \{v\}$. And we have $\mu(\epsilon) = |V|$, because the axioms $\bigcup_{v\in V\setminus u} \alpha(v)$ are satisfiable for all $u \in V$.

(c) If $u \in V'$ and $v \notin V'$, there's an assignment that falsifies C and some axiom of $\alpha(u)$ while satisfying all $\alpha(w)$ for $w \in V' \setminus u$, because $|V'|$ is minimum. Setting $e_{uv} \leftarrow \bar{e}_{uv}$ will satisfy $\alpha(u)$ without affecting the axioms $\alpha(w)$ (which don't contain e_{uv}).

(d) By (b), every refutation of $\alpha(G)$ must contain a clause C with $\frac{1}{3}m \le \mu(C) < \frac{2}{3}m$. The corresponding V' has $|V'|/(|V'| + |\partial V'|) < (\frac{2}{3} + 8)/9$, hence $|\partial V'| > \frac{1}{26}|V'|$.

[Property (i) is interesting but irrelevant for this proof. Notice that $\alpha(G)$ has exactly $\frac{8}{3}n \approx 2.67n$ 3SAT clauses in $n = 3m/2$ variables when G is cubic; every literal occurs four times. G. Tseytin proved lower bounds for refutations of $\alpha(G)$ by *regular* resolution in 1966, before graphs with property (iii) were known; A. Urquhart obtained them for general resolution in *JACM* **34** (1987), 209–219, and the simplified argument given here is due to Ben-Sasson and Wigderson. The fact that $\alpha(G)$ requires exponentially long refutation chains, although the same axioms can be refuted easily by working with linear equations mod 2, amounts to a proof that backtracking is a poor way to deal with linear equations! Suitable Ramanujan graphs $raman(2, q, 3, 0)$ are part of the Stanford GraphBase for infinitely many prime numbers q. We can also obtain the same lower bounds with the multigraphs $raman(2, q, 1, 0)$ and $raman(2, q, 2, 0)$. Section 7.4.3 will explore expander graphs in detail.]

3SAT
cubic
Tseytin
regular resolution
Urquhart
Ben-Sasson
Wigderson
linear equations
backtracking
Stanford GraphBase
raman
multigraphs
expander graphs
merge

246. Let's write $[a_1 \ldots a_k]^\ell$ for what exercise 244 calls $[\{a_1, \ldots, a_k\}]^\ell$. With new variables x, y, z we can introduce $\{xa, x\bar{b}, \bar{x}\bar{a}b, y\bar{a}, yb, \bar{y}a\bar{b}, zx, zy, \bar{z}\bar{x}\bar{y}\}$ and resolve those clauses to $[zab]^1$, which means $z = a \oplus b$. So we can assume that '$z \leftarrow a \oplus b$' is a legal primitive operation of "extended resolution hardware," when z is a new variable. Furthermore we can compute $a_1 \oplus \cdots \oplus a_k$ in $O(k)$ steps, using $z_0 \leftarrow 0$ (which is the clause $[z_0]^1$, namely $\bar{z}_0$) and $z_k \leftarrow z_{k-1} \oplus a_k$ when $k \ge 1$.

Let the edge variables $E(v)$ be $a_1, \ldots, a_d$, where d is the degree of v. We compute $s_v \leftarrow a_1 \oplus \cdots \oplus a_d$ by setting $s_{v,0} \leftarrow 0$, $s_{v,k} \leftarrow s_{v,k-1} \oplus a_k$, and $s_v \leftarrow s_{v,d}$. We can resolve s_v with the axioms $\alpha(v)$ in $O(2^d)$ steps, to get the singleton clause $[s_v]^{\ell(v)\oplus 1}$, meaning $s_v = \ell(v)$. Summing over v, these operations therefore take $O(N)$ steps.

On the other hand, we can also compute $z_n \leftarrow \bigoplus_v s_v$ and get zero (namely '$\bar{z}_n$'). Doing this cleverly, by omnisciently knowing G, we can in fact compute it in $O(mn)$ steps: Start with any vertex v and set $z_1 \leftarrow s_v$ (more precisely, set $z_{1,k} \leftarrow s_{v,k}$ for $0 \le v \le d$). Given z_j for $1 \le j < n$, with all its subvariables $z_{j,k}$, we then compute $z_{j+1} \leftarrow z_j \oplus s_u$, where u is the unused vertex with $s_{u,1} = z_{j,1}$. We can arrange the edges into an order so that if z_j has p edge variables in common with s_u, then $z_{j,k} = s_{u,k}$ for $1 \le k \le p$. Suppose the other variables of z_j and s_u are respectively $a_1, \ldots, a_q$ and $b_1, \ldots, b_r$; we want to merge them into the sequence $c_1, \ldots, c_{q+r}$ that will be needed later when z_{j+1} is used. So we set $z_{j+1,0} \leftarrow 0$, $z_{j+1,k} \leftarrow z_{j+1,k-1} \oplus c_k$, $z_{j+1} \leftarrow z_{j+1,q+r}$.

From the clauses constructed in the previous paragraph, resolution can deduce $[z_{j,k}s_{u,k}]^1$ for $1 \le k \le p$, and hence $[z_{j+1,0}z_{j,p}s_{u,p}]^1$ (namely that $z_{j+1,0} = z_{j,p} \oplus s_{u,p}$). Furthermore, if $c_k = a_i$, and if we know that $z_{j+1,k-1} = z_{j,s} \oplus s_{u,t}$ where $s = p + i - 1$

and $t = p + k - i$, resolution can deduce that $z_{j+1,k} = z_{j,s+1} \oplus s_{u,t}$; a similar formula applies when $c_k = b_i$. Thus resolution yields $z_{j+1} \leftarrow z_j \oplus s_u$ as desired. Ultimately we deduce both z_n and $\bar{z}_n$ from the singleton clauses $s_v = \ell(v)$.

Stålmarck
redundant

247. Eliminating x_2 from $\{12, \bar{1}2, \bar{1}\bar{2}\}$ gives $\{\bar{1}\}$; eliminating x_1 then gives $\emptyset$. So those five clauses are satisfiable.

248. We have $F(x_1, \ldots, x_n) = (x_n \vee A'_1) \wedge \cdots \wedge (x_n \vee A'_p) \wedge (\bar{x}_n \vee A''_1) \wedge \cdots \wedge (\bar{x}_n \vee A''_q) \wedge A'''_1 \wedge \cdots \wedge A'''_r = (x_n \vee G') \wedge (\bar{x}_n \vee G'') \wedge G'''$, where $G' = A'_1 \wedge \cdots \wedge A'_p$, $G'' = A''_1 \wedge \cdots \wedge A''_q$, and $G''' = A'''_1 \wedge \cdots \wedge A'''_r$ depend only on $\{x_1, \ldots, x_{n-1}\}$. Hence $F' = (G' \vee G'') \wedge G'''$; and the clauses of $G' \vee G'' = \bigwedge_{i=1}^{p} \bigwedge_{j=1}^{q} (A'_i \vee A''_j)$ are the resolvents eliminating x_n.

249. After learning $C_7 = \bar{2}\bar{3}$ as in the text, we set $d \leftarrow 2$, $l_2 \leftarrow \bar{2}$, $C_j = 2\bar{3}$, learn $C_8 = \bar{3}$, and set $d \leftarrow 1$, $l_1 \leftarrow \bar{3}$. Then $l_2 \leftarrow \bar{4}$ (say); and $l_3 \leftarrow \bar{1}$, $l_4 \leftarrow \bar{2}$. Now $C_i = 1234$ has been falsified; after $l_4 \leftarrow 2$ and $C_j = 1\bar{2}$ we learn $C_9 = 134$, set $l_3 \leftarrow 1$, and learn $C_{10} = 134 \diamond \bar{1}3 = 34$. Finally $l_2 \leftarrow 4$, we learn $C_{11} = 3$; $l_1 \leftarrow 3$, and we learn $C_{12} = \epsilon$.

250. $l_1 \leftarrow 1$, $l_2 \leftarrow 3$, $l_3 \leftarrow \bar{2}$, $l_4 \leftarrow 4$; learn $\bar{1}2\bar{3}$; $l_3 \leftarrow 2$, $l_4 \leftarrow 4$; learn $\bar{1}\bar{2}\bar{3}$ and $\bar{1}\bar{3}$; $l_2 \leftarrow \bar{3}$, $l_3 \leftarrow \bar{2}$, $l_4 \leftarrow 4$; learn $\bar{1}23$; $l_3 \leftarrow 2$, $l_4 \leftarrow 4$; learn $\bar{1}\bar{2}3$, $\bar{1}3$, $\bar{1}$; $l_1 \leftarrow \bar{1}$, $l_2 \leftarrow 3$, $l_3 \leftarrow \bar{4}$, $l_4 \leftarrow 2$; learn $1\bar{3}4$; $l_3 \leftarrow 4$, $l_4 \leftarrow \bar{2}$, $l_4 \leftarrow 2$.

251. Algorithm I has the property that $\bar{l}_{i_1}, \ldots, \bar{l}_{i_{k-1}}, l_{i_k}$ are on the stack whenever the new clause $l_{i_1} \vee \cdots \vee l_{i_k}$ has been learned, if $i_1 < \cdots < i_k = d$ and step I4 returns to I2. These literals limit our ability to exploit the new clause; so it appears to be impossible to solve this problem without doing more resolutions than Stålmarck did.

However, we can proceed as follows. Let M''_{imk} be the clause $x_{m1} \vee \cdots \vee x_{m(k-1)} \vee x_{ik} \vee \cdots \vee x_{i(m-1)} \vee \bar{x}_{im}$, for $1 \le i, k < m$. Using ij to stand for x_{ij}, the process for $m = 3$ begins by putting $\overline{11}$, $\overline{12}$, 13, $\overline{21}$, $\overline{22}$, 23, $\overline{31}$, $\overline{32}$, 33 on the stack. Then step I3 has $C_i = I_3$, step I4 has $C_j = M_{33}$; so step I5 learns $I_3 \diamond M_{33} = M_{32}$. Step I4 now changes $\overline{32}$ to 32 and chooses $C_j = T_{232}$; so I5 learns $M_{32} \diamond T_{232} = M''_{232}$. Step I4 changes $\overline{31}$ to 31 and chooses $C_j = T_{231}$; now we learn $M''_{232} \diamond T_{231} = M''_{231}$. Next, we learn $M''_{231} \diamond M_{23} = M_{22}$; and after changing $\overline{22}$ to 22 we also learn M_{21}.

The stack now contains $\overline{11}$, $\overline{12}$, 13, 21. We add $\overline{31}$, $\overline{32}$, and proceed to learn $M_{32} \diamond T_{132} = M''_{132}$, $M''_{132} \diamond T_{131} = M''_{131}$, $M''_{131} \diamond M_{13} = M_{12}$. The stack now contains $\overline{11}$, 12, and we've essentially reduced m from 3 to 2.

In a similar way, $O(m^2)$ resolutions will learn $M_{i(m-1)}$ for $i = m-1, \ldots, 1$; and they'll leave $\bar{x}_{11}, \ldots, \bar{x}_{1(m-2)}, x_{1(m-1)}$ on the stack so that the process can continue.

252. No; large numbers of clauses such as $\bar{x}_{12} \vee \bar{x}_{23} \vee \cdots \vee \bar{x}_{89} \vee x_{19}$ are generated by the elimination process. Although these clauses are valid, they're not really helpful.

Exercise 373 proves, however, that the proof *is* completed in polynomial time if we restrict consideration to the transitivity clauses of exercise 228(!).

253. A conflict arises when we follow a chain of forced moves:

t	L_t	level	reason	t	L_t	level	reason
0	$\bar{6}$	1	Λ	5	$\bar{7}$	2	$\bar{5}\bar{7}\bar{9}$
1	4	1	46	6	$\bar{1}$	2	$\bar{1}5\bar{9}$
2	5	2	Λ	7	8	2	678
3	$\bar{3}$	2	$\bar{3}\bar{4}\bar{5}$	8	2	2	123
4	9	2	369	9	$\bar{2}$	2	$\bar{2}5\bar{8}$

Now $\bar{2}5\bar{8} \to \bar{2}5\bar{8} \diamond 123 = 135\bar{8} \to 13\bar{5}67 \to 3\bar{5}67\bar{9} \to 3\bar{5}6\bar{9} \to 3\bar{5}6 \to \bar{4}\bar{5}6$; so we learn $\bar{4}\bar{5}6$ (which can be simplified to $\bar{5}6$, because $\bar{4}$ is "redundant" as explained in exercise 257).

Setting $L_2 \leftarrow \bar{5}$, with reason $\bar{4}\bar{5}6$ or $\bar{5}6$, now forces 7, $\bar{1}$, 3, 9, $\bar{2}$, $\bar{8}$, 8, all at level 1; this conflict soon allows us to learn the *unit* clause 6. (Next we'll inaugurate level 0, setting $L_0 \leftarrow 6$. No "reasons" need to be given at level 0.)

254. Deducing 3, 2, 4, $\bar{4}$ at level 1, it will find $\bar{2}\bar{4} \diamond 4\bar{3} = \bar{2}\bar{3}$ and $\bar{2}\bar{3} \diamond 2\bar{3} = \bar{3}$, learning $\bar{3}$. (Or it might learn $\bar{3}$ after deducing $\bar{2}$.) Then it will deduce $\bar{3}$, $\bar{1}$, 2, $\bar{4}$ at level 0.

255. For example, $\{\bar{1}2\bar{4}, \bar{2}3\bar{5}, 456, 45\bar{6}\}$. [Since the clause c' that is learned by the procedure described in the text contains just one literal l from the conflict level d, the trail position for $\bar{l}$ has been called a "unique implication point" (UIP). If l isn't the decision literal for its level, we could resolve c' with l's reason and find another UIP; but each new resolution potentially increases the b array and limits the amount of backjumping. Therefore we stop at the first UIP.]

256. If it is false, literals 50, 26, ..., 30 are true; hence also $\overline{25}$, 23, and 29, a conflict. Consequently we can obtain '$**$' by starting with $\overline{23}\,\overline{26}\ldots\overline{50}$ and resolving with 23 25 27, 25 27 29, and $\overline{25}\,\overline{30}\ldots\overline{70}$. [Similarly, and more simply, one can learn (122) by resolving $\overline{11}\,\overline{16}\ldots\overline{56}$ with 31 61 91, 41 66 91, and 56 61 66.]

257. (a) Suppose $\bar{l}'$ on level $d' > 0$ is redundant. Then some l'' in the reason for l' is also on level d'; and l'' is either in c or redundant. Use induction on trail position.

(b) We can assume that the stamp value s used when resolving conflicts is a multiple of 3, and that all stamps are $\leq s$. Then we can stamp literal l with $\mathtt{S}(|l|) \leftarrow s+1$ if $\bar{l}$ is known to be redundant, or $s+2$ if $\bar{l}$ is known to be nonredundant and not in c. (These stamps serve as a "memo cache" to avoid repeated work.) While building c we can also stamp *levels* as well as literals, setting $\mathtt{LS}[d'] \leftarrow s$ if level d' has exactly one of the b_i, or $s+1$ if it has more than one.

Then for $1 \leq j \leq r$, $\bar{b}_j$ is redundant if and only if $\mathtt{LS}[\mathit{lev}(b_j)] = s+1$ and $\mathit{red}(\bar{b}_j)$ is true, where $\mathit{lev}(l) = \mathtt{VAL}(|l|) \gg 1$ and where $\mathit{red}(l)$ is the following recursive procedure: "If l is a decision literal, return false. Otherwise let $(l \vee \bar{a}_1 \vee \cdots \vee \bar{a}_k)$ be l's reason. For $1 \leq i \leq k$ with $\mathit{lev}(a_i) > 0$, if $\mathtt{S}(|a_i|) = s+2$ return false; if $\mathtt{S}(|a_i|) < s$ and either $\mathtt{LS}[\mathit{lev}(a_i)] < s$ or $\mathit{red}(\bar{a}_i)$ is false, set $\mathtt{S}(|a_i|) \leftarrow s+2$ and return false. But if none of these conditions hold, set $\mathtt{S}(|l|) \leftarrow s+1$ and return true."

[See Allen Van Gelder, *LNCS* **5584** (2009), 141–146.]

258. That statement is true in Table 3, but false in general. Indeed, consider the sequel to Table 3: The decision $L_{44} = \overline{57}$ causes the watch list of 57 to be examined, thus forcing 15, 78, and 87 (among other literals) in some order because of the clauses 15 57 36, 78 57 36, 87 57 27. Then $\overline{96}$ will be forced by the clause $\overline{96}\,\overline{87}\ldots\overline{15}$; and the second literal of that clause at the time of forcing will be $\overline{15}$, regardless of trail order, if the watched literals of that clause were $\overline{96}$ and $\overline{15}$ (making it invisible to $\overline{78}$ and $\overline{87}$).

259. $1+\rho^6+\rho^7 < \rho+\rho^2$ when $.7245 < \rho < .7548$. (There can in fact be any number of crossover points: Consider the polynomial $(1-\rho-\rho^2)(1-\rho^3-\rho^6)(1-\rho^9-\rho^{18})$.)

260. First, to get a random permutation in the heap we can use a variant of Algorithm 3.4.2P: For $k \leftarrow 1, 2, \ldots, n$, let j be a random integer in $[0\,..\,k-1]$ and set $\mathtt{HEAP}[k-1] \leftarrow \mathtt{HEAP}[j]$, $\mathtt{HEAP}[j] \leftarrow k$. Then set $\mathtt{HLOC}(\mathtt{HEAP}[j]) \leftarrow j$ for $0 \leq j < n$.

Next, set $F \leftarrow 0$ and $W_l \leftarrow 0$ for $2 \leq l \leq 2n+1$ and $c \leftarrow 3$. Do the following for each input clause $l_0 l_1 \ldots l_{k-1}$: Terminate unsuccessfully if $k=0$, or if $k=1$ and $0 \leq \mathtt{VAL}(|l_0|) \neq l_0 \,\&\, 1$. If $k=1$ and $\mathtt{VAL}(|l_0|) < 0$, set $\mathtt{VAL}(|l_0|) \leftarrow l_0 \,\&\, 1$, $\mathtt{TLOC}(|l_0|) \leftarrow F$, $F \leftarrow F+1$. If $k > 1$, set $\mathtt{MEM}[c+j] \leftarrow l_j$ for $0 \leq j < k$; also $\mathtt{MEM}[c-1] \leftarrow k$, $\mathtt{MEM}[c-2] \leftarrow W_{l_0}$, $W_{l_0} \leftarrow c$, $\mathtt{MEM}[c-3] \leftarrow W_{l_1}$, $W_{l_1} \leftarrow c$, $c \leftarrow c+k+3$.

level 0
reasons
unique implication point
UIP
backjumping
memo cache
memoization technique
lev
recursive procedure
Van Gelder
watched literals
random permutation

Finally, set MINL $\leftarrow$ MAXL $\leftarrow c+2$ (allowing two cells for extra data in the preamble of the first learned clause). Of course we must also ensure that MEM is large enough.

watched literals
lazy data structures
heap deletion
deletion from heap
siftup
heap insertion
insertion into a heap
blit
redundancies
watch a literal
flushed
restarted

261. (Throughout this answer, l_j is an abbreviation for MEM[$c+j$].) Set $q \leftarrow 0$ and $c \leftarrow W_{\bar l}$. While $c \neq 0$, do the following: Set $l' \leftarrow l_0$. If $l' \neq \bar l$ (hence $l_1 = \bar l$), set $c' \leftarrow l_{-3}$; otherwise set $l' \leftarrow l_1$, $l_0 \leftarrow l'$, $l_1 \leftarrow \bar l$, $c' \leftarrow l_{-2}$, $l_{-2} \leftarrow l_{-3}$, and $l_{-3} \leftarrow c'$. If VAL($|l_0|$) ≥ 0 and VAL($|l_0|$) $+ l_0$ is even (that is, if l_0 is true), perform the steps

$$\text{if } q \neq 0, \text{ set } \texttt{MEM[}q-3\texttt{]} \leftarrow c, \text{ else set } W_{\bar l} \leftarrow c; \text{ then set } q \leftarrow c. \qquad (*)$$

Otherwise set $j \leftarrow 2$; while $j < l_{-1}$ and VAL($|l_j|$) ≥ 0 and VAL($|l_j|$) $+ l_j$ is odd, set $j \leftarrow j+1$. If now $j < l_{-1}$, set $l_1 \leftarrow l_j$, $l_j \leftarrow \bar l$, $l_{-3} \leftarrow W_{l_1}$, $W_{l_1} \leftarrow c$. But if $j = l_{-1}$, do $(*)$ above; jump to C7 if VAL($|l_0|$) ≥ 0; otherwise set $L_F \leftarrow l_0$, etc. (see step C4) and $c \leftarrow c'$.

Finally, when $c = 0$, do $(*)$ above to terminate $\bar l$'s new watch list.

262. To delete $k =$ HEAP[0] in C6: Set $h \leftarrow h-1$ and HLOC(k) $\leftarrow -1$. Stop if $h = 0$. Otherwise set $i \leftarrow$ HEAP[h], $\alpha \leftarrow$ ACT(i), $j \leftarrow 0$, $j' \leftarrow 1$, and do the following while $j' < h$: Set $\alpha' \leftarrow$ ACT(HEAP[j']); if $j'+1 < h$ and ACT(HEAP[$j'+1$]) $> \alpha'$, set $j' \leftarrow j'+1$ and $\alpha' \leftarrow$ ACT(HEAP[j']); if $\alpha \geq \alpha'$, set $j' \leftarrow h$, otherwise set HEAP[j] $\leftarrow$ HEAP[j'], HLOC(HEAP[j']) $\leftarrow j$, $j \leftarrow j'$, and $j' \leftarrow 2j+1$. Then set HEAP[j] $\leftarrow i$ and HLOC(i) $\leftarrow j$.

In C7, set $k \leftarrow |l|$, $\alpha \leftarrow$ ACT(k), ACT(k) $\leftarrow \alpha +$ DEL, $j \leftarrow$ HLOC(k), and if $j > 0$ perform the "siftup" operation: "Looping repeatedly, set $j' \leftarrow (j-1) \gg 1$ and $i \leftarrow$ HEAP[j'], exit if ACT(i) $\geq \alpha$, else set HEAP[j] $\leftarrow i$, HLOC(i) $\leftarrow j$, $j \leftarrow j'$, and exit if $j = 0$. Then set HEAP[j] $\leftarrow k$ and HLOC(k) $\leftarrow j$."

To insert k in C8, set $\alpha \leftarrow$ ACT(k), $j \leftarrow h$, $h \leftarrow h+1$; if $j = 0$ set HEAP[0] $\leftarrow k$ and HLOC(k) $\leftarrow 0$; otherwise perform the siftup operation.

263. (This answer also sets the level stamps LS[d] needed in answer 257, assuming that the LS array is initially zero.) Let "bump l" mean "increase ACT($|l|$) by DEL" as in answer 262. Also let $\mathit{blit}(l)$ be the following subroutine: "If S($|l|$) $= s$, do nothing. Otherwise set S($|l|$) $\leftarrow s$, $p \leftarrow \mathit{lev}(l)$. If $p > 0$, bump l; then if $p = d$, set $q \leftarrow q+1$; else set $r \leftarrow r+1$, $b_r \leftarrow \bar l$, $d' \leftarrow \max(d', p)$, and if LS[$p$] $\leq s$ set LS[p] $\leftarrow s + [\texttt{LS[}p\texttt{]} = s]$."

When step C7 is entered from C4, assuming that $d > 0$, set $d' \leftarrow q \leftarrow r \leftarrow 0$, $s \leftarrow s+3$, S($|l_0|$) $\leftarrow s$, bump l_0, and do $\mathit{blit}(l_j)$ for $1 \leq j < k$. Also set $t \leftarrow \max(\texttt{TLOC(}|l_1|\texttt{)}, \ldots, \texttt{TLOC(}|l_{k-1}|\texttt{)})$. Then, while $q > 0$, set $l \leftarrow L_t$, $t \leftarrow t-1$; if S($|l|$) $= s$ then set $q \leftarrow q-1$, and if $R_l \neq \Lambda$ let clause R_l be $l_0 l_1 \ldots l_{k-1}$ and do $\mathit{blit}(l_j)$ for $1 \leq j < k$. Finally set $l' \leftarrow L_t$, and while S($|l'|$) $\neq s$ set $t \leftarrow t-1$ and $l' \leftarrow L_t$.

The new clause can now be checked for redundancies as in answer 257. To install it during step C9, there's a subtle point: *We must watch a literal that was defined on level d'.* Thus we set $c \leftarrow$ MAXL, MEM[c] $\leftarrow \bar l'$, $k \leftarrow 0$, $j' \leftarrow 1$; and for $1 \leq j \leq r$ if S($|b_j|$) $= s$ set $k \leftarrow k+1$ and do this: If $j' = 0$ or $\mathit{lev}(|b_j|) < d'$, set MEM[$c+k+j'$] $\leftarrow \bar b_j$, otherwise set MEM[$c+1$] $\leftarrow \bar b_j$, $j' \leftarrow 0$, MEM[$c-2$] $\leftarrow W_{\bar l'}$, $W_{\bar l'} \leftarrow c$, MEM[$c-3$] $\leftarrow W_{\bar b_j}$, $W_{\bar b_j} \leftarrow c$. Finally set MEM[$c-1$] $\leftarrow k+1$, MAXL $\leftarrow c+k+6$.

264. We can maintain a "history code" array, setting H_F to 0, 2, 4, or 6 when L_F is set, and then using $H_t + (L_t \mathbin{\&} 1)$ as the move code that represents trail location t for $0 \leq t < F$. History codes 6, 4, and 0 are appropriate in steps C1, C4, and C6, respectively; in C9, use code 2 if l' was a decision literal, otherwise use code 6.

[These move codes do *not* increase lexicographically when the trail is flushed and restarted; hence they don't reveal progress as nicely as they do in the other algorithms.]

265. (1) A literal L_t on the trail with $G \leq t < F$ has become true, but the watch list of $\overline{L_t}$ has not yet been examined. (2) If l_0 is true, so that c is satisfied, step C4 doesn't

remove c from the watch list of l_1 when l_1 becomes false. (This behavior is justified, because c won't be examined again until l_1 has become free during the backtracking step C8.) (3) A clause that becomes a reason for l_0 remains on the watch list of its false l_1. (4) During a full run, a clause that triggers a conflict is allowed to keep both of its watched literals false.

In general, a false watched literal must be defined at the highest trail level of all literals in its clause.

full run
BIMP(l)
Knuth
buddy-system
breadth-first
flag
purging
watched literals

266. If $U < p$, where U is a uniform deviate between 0 and 1, do this: Set j to a random integer with $0 \le j < h$, and $k \leftarrow$ HEAP[j]. If $j = 0$, or if VAL(k) ≥ 0, use the normal C6. Otherwise branch on k (and don't bother to remove k from the heap).

267. As in Algorithm L, let there be a sequential table BIMP(l) for each literal l, containing all literals l' such that $\bar{l} \vee l'$ is a binary clause. Furthermore, when the propagation algorithm sets $L_F \leftarrow l'$ because $l' \in$ BIMP(l), we may set $R_{l'} \leftarrow -l$, instead of using a positive clause number as the "reason." (Notice that a binary clause therefore need not be represented explicitly in MEM, if it is represented implicitly in the BIMP tables. The author's implementation of Algorithm C uses BIMP tables only to expedite binary clauses that appear in the original input. This has the advantage of simplicity, since the exact amount of necessary space can be allocated permanently for each table. Learned binary clauses are comparatively rare in practice; thus they can usually be handled satisfactorily with watched literals, instead of by providing the elaborate buddy-system scheme that was important in Algorithm L.)

Here, more precisely, is how the inner loop goes faster with BIMPs. We want to carry out binary propagations as soon as possible, because of their speed; hence we introduce a breadth-first exploration process analogous to (62):

$$\begin{array}{l}\text{Set } H \leftarrow F\text{; take account of } l' \text{ for all } l' \in \texttt{BIMP}(l_0)\text{;}\\ \text{while } H < F\text{, set } l_0 \leftarrow R_H,\ H \leftarrow H+1\text{, and}\\ \qquad \text{take account of } l' \text{ for all } l' \in \texttt{BIMP}(l_0)\text{.}\end{array} \qquad (**)$$

Now "take account of l'" means "if l' is true, do nothing; if l' is false, go to C7 with conflict clause $\bar{l} \vee l'$; otherwise set $L_F \leftarrow l'$, TLOC($|l'|$) $\leftarrow F$, VAL($|l'|$) $\leftarrow 2d + (l' \,\&\, 1)$, $R_{l'} \leftarrow -l$, $F \leftarrow F+1$." We do (∗∗) just before setting $c \leftarrow c'$ in answer 261. Furthermore, we set $E \leftarrow F$ just after $G \leftarrow 0$ in step C1 and just after $F \leftarrow F+1$ in steps C6 and C9; and if $G \le E$ after $G \leftarrow G+1$ in step C4, we do (∗∗) with $l_0 \leftarrow \bar{l}$.

Answer 263 is modified in straightforward ways so that "clause R_l" is treated as if it were the binary clause $(l \vee \bar{l}')$ when R_l has the negative value $-l'$.

268. If MEM[$c-1$] $= k \ge 3$ is the size of clause c, and if $1 < j < k$, we can delete the literal l in MEM[$c+j$] by setting $k \leftarrow k-1$, MEM[$c-1$] $\leftarrow k$, $l' \leftarrow$ MEM[$c+k$], MEM[$c+j$] $\leftarrow l'$, and MEM[$c+k$] $\leftarrow l+f$, where f is a flag (typically 2^{31}) that distinguishes a deleted literal from a normal one. (This operation does not need to be done when the current level d is zero; hence we can assume that $k \ge 3$ and $j > 1$ before deletion. The flag is necessary so that global operations on the entire set of clauses, such as the purging algorithm, can pass safely over deleted literals. The final clause in MEM should be followed by 0, an element that's known to be unflagged.)

269. (a) If the current clause contains a literal $l = \bar{L}_t$ that is not in the trivial clause, where t is maximum, resolve the current clause with $R_{\bar{l}}$ and repeat.

(b) $(\bar{u}_1 \vee b_j) \wedge (l_j \vee \bar{l}_{j-1} \vee \bar{b}_j)$ for $1 \le j \le 9$, $(l_0 \vee \bar{u}_2 \vee \bar{u}_3) \wedge (\bar{l}_9 \vee \bar{l}_8 \vee \bar{b}_{10})$; $l' = l_0$.

(c) If $r \ge d' + \tau$, where τ is a positive parameter, learn the trivial clause instead of $(\bar{l}' \vee \bar{b}_1 \vee \cdots \vee \bar{b}_r)$. (The watched literals should be $\bar{l}'$ and $\bar{u}_{d'}$.)

Notice that this procedure will learn more than simple backtrack à la Algorithm D does, even when the trivial clause is *always* substituted (that is, even when $\tau = -\infty$), because it provides for backjumping when $d' < d + 1$.

270. (a) Consider the clauses $3\bar{2}$, $4\bar{3}\bar{2}$, $5\bar{4}\bar{3}\bar{1}$, $6\bar{5}\bar{4}\bar{1}$, $\bar{6}\bar{5}\bar{4}$, with initial decisions $L_1 \leftarrow 1$, $L_2 \leftarrow 2$. Then $L_3 \leftarrow 3$ with reason $R_3 \leftarrow 3\bar{2}$; similarly $L_4 \leftarrow 4$, $L_5 \leftarrow 5$. If $L_6 \leftarrow 6$, the conflict clause $\bar{6}\bar{5}\bar{4}$ allows us to strengthen R_6 to $\bar{5}\bar{4}\bar{1}$; but if $L_6 \leftarrow \bar{6}$, with $R_{\bar{6}} \leftarrow \bar{6}\bar{5}\bar{4}$, we don't notice that $6\bar{5}\bar{4}\bar{1}$ can be strengthened. In either case we can, however, strengthen R_5 to $\bar{4}\bar{3}\bar{1}$, before learning the clause $\bar{2}\bar{1}$.

(b) After doing $blit(l_j)$ to the literals of R_l, we know that $R_l \setminus l$ is contained in $\{\bar{b}_1, \ldots, \bar{b}_r\}$ together with $q+1$ unresolved false literals that have been stamped at level d. (Exercise 268 ensures that $p \neq 0$ within each *blit*.) Thus we can subsume clause R_l on the fly if $q + r + 1 < k$ and $q > 0$.

In such cases the procedure of answer 268 can be used to delete l from $c = R_l$. But there's a complication, because $l = l_0$ is a watched literal ($j = 0$ in that answer), and all other literals are false. After l is deleted, it will be essential to watch a false literal l' that is defined at trail level d. So we find the largest $j' \le k$ such that $\texttt{VAL(MEM[}c + j'\texttt{])} \ge 2d$, and we set $l' \leftarrow \texttt{MEM[}c + j'\texttt{]}$. If $j' \neq k$, we also set $\texttt{MEM[}c + j'\texttt{]} \leftarrow \texttt{MEM[}c + k\texttt{]}$; we can assume that $j' > 1$. Finally, after setting $\texttt{MEM[}c\texttt{]} \leftarrow l'$ and $\texttt{MEM[}c + k\texttt{]} \leftarrow l + f$ as in answer 268, we also delete c from the watch list W_l, and insert it into $W_{l'}$.

[This enhancement typically saves 1%–10% of the running time, but sometimes it saves a lot more. It was discovered in 2009, independently by two different groups of researchers: See H. Han and F. Somenzi, *LNCS* **5584** (2009), 209–222; Y. Hamadi, S. Jabbour, and L. Saïs, *Int. Conf. Tools with Artif. Int.* (ICTAI) **21** (2009), 328–335.]

271. We shall check for discards only if the current clause C_i is not trivial (see exercise 269), and if the first literal of C_{i-1} does not appear in the trail. (Indeed, experience shows that almost every permissible discard falls into this category.) Thus, let C_{i-1} be $l_0 l_1 \ldots l_{k-1}$ where $\texttt{VAL(}|l_0|\texttt{)} < 0$; we want to decide if $\{\bar{l}', \bar{b}_1, \ldots, \bar{b}_r\} \subseteq \{l_1, \ldots, l_{k-1}\}$.

The secret is to use the stamp fields that have already been set up. Set $j \leftarrow k-1$, $q \leftarrow r+1$, and do the following while $q > 0$ and $j \ge q$: If $l_j = \bar{l}'$, or if $\texttt{VAL(}|l_j|\texttt{)} \le 2d'+1$ and $\texttt{S(}|l_j|\texttt{)} = s$, set $q \leftarrow q - 1$; in any case set $j \leftarrow j - 1$. Then discard if $q = 0$.

272. Reflection isn't as easy to implement as it may seem, unless C is a unit clause, because C^R must be placed carefully in MEM and it must be consistent with the trail. Furthermore, experience shows that it's best not to learn the reflection of *every* learned clause, because excess clauses make unit propagation slower. The author has obtained encouraging results, however, by doing the following operations just before returning to C3 in step C9, whenever the length of C doesn't exceed a given parameter R:

Assign ranks to the literals of C^R by letting $\mathrm{rank}(l) = \infty$ if l is on the trail, $\mathrm{rank}(l) = d''$ if $\bar{l}$ is on the trail at level $d'' < d'$, $\mathrm{rank}(l) = d$ otherwise. Let u and v be two of the highest ranking literals, with $\mathrm{rank}(u) \ge \mathrm{rank}(v)$. Put them into the first two positions of C^R, so that they will be watched. Do nothing further if $\mathrm{rank}(v) > d'$. Otherwise, if $\mathrm{rank}(v) < d'$, backjump to level $\mathrm{rank}(v)$ and set $d' \leftarrow \mathrm{rank}(v)$. Then if $\mathrm{rank}(u) = \mathrm{rank}(v) = d'$, treat C^R as a conflict clause by going to step C7 with $c \leftarrow C^R$. (That is a rare event, but it can happen.) Otherwise, if u doesn't appear in the current trail, set $L_F \leftarrow u$, $\texttt{TLOC(}|u|\texttt{)} \leftarrow F$, $R_u \leftarrow C^R$, $F \leftarrow F + 1$. (Possibly $F = E + 2$ now.)

(For example, this method with $R \leftarrow 6$ roughly halved the running time of *waerden*$(3, 10; 97)$ and *waerden*$(3, 13; 160)$ with parameters (193) except for $\rho \leftarrow .995$.)

A similar idea works with the clauses *langford*(n), and in general whenever the input clauses have an automorphism of order 2.

backtrack
backjumping
blit
watched literal
Han
Somenzi
Hamadi
Jabbour
Saïs
trivial
stamp
trail
unit propagation
author
backjump
langford(n)
automorphism

273. (a) We can convert Algorithm C into a "clause learning machine" by keeping the process going after F reaches n in step C5: Instead of terminating, start over again by essentially going back to step C1, except that the current collection of clauses should be retained, and the `OVAL` polarities should be reset to random bits. Learned clauses of size K or less, where K is a parameter, should be written to a file. Stop when you've found a given number of short clauses, or when you've exceeded a given time limit.

For example, here's what happened when the author first tried to find $W(3,13)$: Applying this algorithm to $waerden(3,13;158)$ with $K=3$, and with a timeout limit of 30 Gμ (gigamems), yielded the five clauses 65 68 70, 68 78 81, 78 81 90, 78 79 81, 79 81 82. So fifteen clauses 65 68 70, 66 69 71, ..., 81 83 84 could be added to $waerden(3,13;160)$, as well as their fifteen reflections 96 93 91, 95 92 90, ..., 80 78 77. Then the algorithm "$\mathrm{C^R}$" of exercise 272 proved this augmented set unsatisfiable after an additional 107 Gμ. In a second experiment, using $K=2$ with $waerden(3,13;159)$ led to three binary clauses 76 84, 81 86, and 84 88. Shifting and reflecting gave twelve binary clauses, which in company with $waerden(3,13;160)$ were refuted by $\mathrm{C^R}$ in another 80 Gμ. (For comparison, Algorithm $\mathrm{C^R}$ refuted $waerden(3,13;160)$ unaided in about 120 Gμ, compared to about 270 Gμ for both Algorithm C and Algorithm L.) Optimum strategies for learning useful clauses from satisfiable subproblems are far from clear, especially because running times are highly variable. But this method does show promise, especially on more difficult problems — when more time can be devoted to the preliminary learning.

(b) Short clauses that can be learned from satisfiable instances of, say, $X_0 \to X_1 \to \cdots \to X_{r-1}$, when X_0 is *not* required to be an initial state, can be shifted and used to help refute $X_0 \to X_1 \to \cdots \to X_r$.

274. With care, circular reasoning can (and must) be avoided. But the author's elaborate experiments with such ideas (and with the related notion of "better conflicts") were disappointing; they didn't beat the running time of the simpler algorithm. However, an intriguing idea by Allen Van Gelder [*Journal on Satisfiability, Boolean Modeling and Computation* **8** (2012), 117–122] shows promise.

275. When a solution has been found, let k be minimum such that $x_k = 1$ and the value of x_k has not been assigned at level 0. If no such k exists, we stop. Otherwise we are entitled to force variables x_1 through x_{k-1} all to have their current values, at level 0, because we know that this doesn't produce an unsatisfiable problem. So we fix those values, and we restart the solution process at level 1 with the tentative decision '$x_k = 0$'. If a conflict occurs, we'll know that $x_k = 1$ at level 0; if not, we'll have a solution with $x_k = 0$. In either case we can increase k. (This method is considerably better than that of answer 109, because *every learned clause remains valid.*)

276. True. Unit propagation essentially transforms $F \wedge L$ into $F \,|\, L$.

277. Otherwise $F \wedge C_1 \wedge \cdots \wedge C_{t-1} \vdash_1 \epsilon$ fails (unit propagation wouldn't start).

278. For example, $(46, \bar{5}6, \bar{5}4, 6, 4, \epsilon)$. (Six steps are necessary.)

279. True, because the dependency digraph contains a literal l with $l \longrightarrow^* \bar{l} \longrightarrow^* l$.

280. (a) They're satisfied if and only if $x_1 \ldots x_n$ has at least j 0s and at least k 1s. [The problem $cook(k,k)$ was introduced by Stephen A. Cook (unpublished) in 1971.]

(b) Take all positive $(j-t)$-clauses on $\{1,\ldots,n-1-t\}$ for $t = 1, 2, \ldots, j$.

(c) Suppose the very first decision is $L_0 \leftarrow x_n$. The algorithm will proceed to act as if the input were $cook(j,k) \,|\, x_n = cook(j,k-1)$. Furthermore, with these clauses, every clause that it learns initially will include $\bar{x}_n$. Therefore, by induction, the unit clause $(\bar{x}_n)$ will be learned clause number $\binom{n-2}{j-1}$. All previously learned clauses are

OVAL
polarities
author
Knuth
Van Gelder
dependency digraph
Cook

Stålmarck
exclusion clauses
at-most-one
symmetry-breaking
unit clause
data structures

subsumed by this one, hence they're no longer relevant. The remaining problem is $cook(j,k)\,|\,\bar{x}_n = cook(j-1,k)$; so the algorithm will finish after learning $\binom{n-2}{j-2}$ more.

Similarly, if the first decision is $L_0 \leftarrow \bar{x}_n$, the $\binom{n-2}{j-2}$th learned clause will be (x_n).

281. Stålmarck's refutation corresponds to the sequence $(M'_{jk1},\ M'_{jk2},\ \ldots,\ M'_{jk(k-1)},\ M_{j(k-1)})$ for $j = 1, \ldots, k-1$, for $k = m, m-1, \ldots, 1$. ($M'_{jk(k-1)}$ can be omitted.)

282. First learn the exclusion clauses (17). In the next clauses we shall write $a_j, b_j, \ldots,$ as shorthand for $a_{j,p}, b_{j,p}, \ldots,$ where p is a particular color, $1 \le p \le 3$. Notice that the $12q$ edges appear in $4q$ triangles, namely $\{b_j, c_j, d_j\}$, $\{a_j, a_{j'}, b_{j'}\}$, $\{f_j, e_{j'}, c_{j'}\}$, $\{e_j, f_{j'}, d_{j'}\}$, for $1 \le j \le q$, where j' is $j+1$ (modulo q). For every such triangle $\{u, v, w\}$, learn $(\bar{u}_{p'} \vee v_p \vee w_p)$ and then $(u_p \vee v_p \vee w_p)$, where p' is $p+1$ (modulo 3).

Now for $j = 1, 2, \ldots, q$, learn $(a_j \vee f_j \vee a_{j'} \vee e_{j'})$, $(a_j \vee e_j \vee a_{j'} \vee f_{j'})$, $(e_j \vee f_j \vee e_{j'} \vee f_{j'})$, $(\bar{a}_j \vee \bar{e}_j \vee \bar{e}_{j'})$, $(\bar{a}_j \vee \bar{f}_j \vee \bar{f}_{j'})$, $(\bar{e}_j \vee \bar{f}_j \vee \bar{a}_{j'})$, as well as eighteen more:

$$(\bar{u}_1 \vee \bar{v}_1 \vee u'_j \vee v'_j), (\bar{u}_2 \vee \bar{v}_2 \vee u'_{j'} \vee v'_{j'}), \quad \text{if } j \ge 3 \text{ is odd;}$$
$$(\bar{u}_1 \vee \bar{v}_1 \vee \bar{u}'_j), (\bar{u}_2 \vee \bar{v}_2 \vee \bar{u}'_{j'}), \quad \text{if } j \ge 3 \text{ is even;}$$

here $u, v \in \{a, e, f\}$ and $u', v' \in \{a, e, f\}$ yield 3×3 choices of (u, v, u', v'). Then we're ready to learn $(\bar{a}_j \vee \bar{e}_j)$, $(\bar{a}_j \vee \bar{f}_j)$, $(\bar{e}_j \vee \bar{f}_j)$ for $j \in \{1, 2\}$ and $(a_j \vee e_j \vee f_j \vee a_{j'})$, $(a_j \vee e_j \vee f_j)$ for $j \in \{1, q\}$. All of these clauses are to be learned for $1 \le p \le 3$.

Next, for $j = q, q-1, \ldots, 2$, learn $(\bar{a}_j \vee \bar{e}_j)$, $(\bar{a}_j \vee \bar{f}_j)$, $(\bar{e}_j \vee \bar{f}_j)$ for $1 \le p \le 3$ and then $(a_{j-1} \vee e_{j-1} \vee f_{j-1} \vee a_j)$, $(a_{j-1} \vee e_{j-1} \vee f_{j-1})$ for $1 \le p \le 3$. We have now established all clauses in the hint.

The endgame consists of the following for $1 \le p \le 3$: For all choices of p' and p'' with $\{p, p', p''\} = \{1, 2, 3\}$ (thus two choices), and for $j = 2, 3, \ldots, q$, learn three clauses

$$(\bar{a}_{1,p} \vee \bar{e}_{1,p'} \vee \bar{a}_{j,p} \vee e_{j,p''}), (\bar{a}_{1,p} \vee \bar{e}_{1,p'} \vee \bar{a}_{j,p'} \vee e_{j,p}), (\bar{a}_{1,p} \vee \bar{e}_{1,p'} \vee \bar{a}_{j,p''} \vee e_{j,p'}),\ \ j \text{ even;}$$
$$(\bar{a}_{1,p} \vee \bar{e}_{1,p'} \vee \bar{a}_{j,p} \vee e_{j,p'}), (\bar{a}_{1,p} \vee \bar{e}_{1,p'} \vee \bar{a}_{j,p'} \vee e_{j,p''}), (\bar{a}_{1,p} \vee \bar{e}_{1,p'} \vee \bar{a}_{j,p''} \vee e_{j,p}),\ \ j \text{ odd;}$$

then learn $(\bar{a}_{1,p} \vee \bar{e}_{1,p'})$. Finally learn $\bar{a}_{1,p}$.

[Not all of these clauses are actually necessary. For example, the exclusion clauses for b's, c's, and d's aren't used. This certificate doesn't assume that the symmetry-breaking unit clauses $b_{1,1} \wedge c_{1,2} \wedge d_{1,3}$ of *fsnark*(q) are present; indeed, those clauses don't help it much. The actual clauses learned by Algorithm C are considerably longer and somewhat chaotic (indeed mysterious); it's hard to see just where an "aha" occurs!]

283. A related question is to ask whether the expected length of learned clauses is $O(1)$ as $q \to \infty$.

284. It's convenient to represent each unit clause (l) in $F \cup C_1 \cup \cdots \cup C_t$ as if it were the binary clause $(l \vee \bar{x}_0)$, where x_0 is a new variable that is always true. We borrow some of the data structures of Algorithm C, namely the trail array L, the reason array R, and the fields `TLOC`, `S`, `VAL` associated with each variable. We set `VAL`$(k) = 0$, 1, or -1 when x_k has been forced true, forced false, or not forced, respectively.

To verify the clause $C_i = (a_1 \vee \cdots \vee a_k)$, we begin with `VAL`$(j) \leftarrow 0$ for $0 \le j \le n$, $L_0 \leftarrow 0$, $L_1 \leftarrow \bar{a}_1, \ldots, L_k \leftarrow \bar{a}_k$, $E \leftarrow F \leftarrow k+1$, $G \leftarrow 0$, and `VAL`$(|L_p|) \leftarrow L_p \,\&\, 1$ for $0 \le p < F$; then we carry out unit propagation as in Algorithm C, expecting to reach a conflict before $G = F$. (Otherwise verification fails.)

A conflict arises when a clause $c = l_0 \ldots l_{k-1}$ forces l_0 at a time when $\bar{l}_0$ has already been forced. Now we mimic step C7 (see exercise 263), but the operations are much simpler: Mark c, stamp `S`$(|l_j|) \leftarrow i$ for $0 \le j < k$, and set $p \leftarrow \max($`TLOC`$(|l_1|), \ldots,$

TLOC($|l_{k-1}|$)). Now, while $p \geq E$, we set $l \leftarrow L_p$, $p \leftarrow p - 1$, and if S($|l|$) $= i$ we also "resolve with the reason of l" as follows: Let clause R_l be $l_0 l_1 \ldots l_{k-1}$, mark R_l, and set S($|l_j|$) $\leftarrow i$ for $1 \leq j < k$.

[Wetzler, Heule, and Hunt have suggested an interesting improvement, which will often mark significantly fewer clauses at the expense of a more complicated algorithm: Give preference to already-marked clauses when doing the unit propagations, just as Algorithm L prefers binary implications to the implications of longer clauses (see (62)).]

Wetzler
Heule
Hunt
worst case
tie-breakers
clause activity
trivial
backjump
floating point

285. (a) $j = 77$, $s_{77} = 12 + 2827$, $m_{77} = 59$, $b_{77} = 710$.

(b) $j = 72$, $s_{72} = 12 + 2048$, $m_{72} = 99 + 243 + 404 + 536 = 1282$, $b_{72} = 3 + 40 + 57 + 86 = 186$. (The RANGE statistic is rather coarse when $\alpha = \frac{1}{2}$, because many different signatures yield the same value.)

(c) $j = 71$, $s_{71} = 12 + 3087$, $m_{71} = 243$, $b_{71} = 40$.

286. The maximum, 738, is achieved uniquely by the RANGE-oriented solution with $\alpha = \frac{15}{16}$, except that we can optionally include also the signatures $(6,0)$ and $(7,0)$ for which $a_{pq} = 0$. [This solution optimizes the worst case of clause selection, because the stated problem implicitly assumes that the secondary heuristic is bad. If we assume, however, that the choice of tie-breakers based on clause activity is at least as good as a random choice, then the expected number $738 + 45 \cdot \frac{10}{59} \approx 745.6$ from $\alpha = \frac{15}{16}$ is *not* as good as the expected number $710 + 287 \cdot \frac{57}{404} \approx 750.5$ from $\alpha = \frac{9}{16}$.]

287. When a conflict is detected in step C7 (with $d > 0$), keep going as in step C3; but remember the first clause C_d that detected a conflict at each level d.

Eventually step C5 will find $F = n$. That's when clauses get their RANGE scores, if we're doing a full run because we want to purge some of them. (Sometimes, however, it's also useful to do a few full runs at the very beginning, or just after a restart, because some valuable clauses might be learned.)

New clauses can be learned in the usual way from the remembered clauses C_d, in decreasing order of d, except that "trivial" clauses (exercise 269) are considered only at the lowest such level. We must keep track of the minimum backjump level d', among all of these conflicts. And if several new clauses have the same d', we must remember all of the literals that will be placed at the end of the trail after we eventually jump back.

288. Step C5 initiates a full run, then eventually finds $F = n$. At this point we're done, in the unlikely event that no conflicts have arisen. Otherwise we set LS[d] $\leftarrow 0$ for $0 \leq d < n$ and $m_j \leftarrow 0$ for $1 \leq j < 256$. The activity ACT(c) of each learned clause c has been maintained in MEM[$c - 5$], as a 32-bit floating point number. The following steps compute RANGE(c), which will be stored in MEM[$c - 4$] as an integer, for all learned c in increasing order, assuming that c's literals are $l_0 l_1 \ldots l_{s-1}$:

If $R_{l_0} = c$, set RANGE(c) $\leftarrow 0$. Otherwise set $p \leftarrow r \leftarrow 0$, and do the following for $0 \leq k < s$: If $v < 2$ and $v + l_k$ is even, set RANGE(c) $\leftarrow 256$ and exit the loop on k (because c is permanently satisfied, hence useless). If $v \geq 2$ and LS[$lev(l_k)$] $< c$, set LS[$lev(l_k)$] $\leftarrow c$ and $r \leftarrow r+1$. Then if $v \geq 2$ and LS[$lev(l_k)$] $= c$ and $l_k + v$ is even, set LS[$lev(l_k)$] $\leftarrow c + 1$ and $p \leftarrow p + 1$. After k reaches s, set $r \leftarrow \min(\lfloor 16(p + \alpha(r - p)) \rfloor, 255)$, RANGE($c$) $\leftarrow r$, and $m_r \leftarrow m_r + 1$.

Now resolve conflicts (see answer 287), giving ACT(c) $\leftarrow 0$ and RANGE(c) $\leftarrow 0$ to all newly learned clauses c, and jump back to trail level 0. (A round of purging is a major event, something like spring cleaning. It is possible that $d' = 0$, in which case one or more literals have been appended to trail level 0 and their consequences have not yet been explored.) Find the median range j as defined in (124), where T is half the total current number of learned clauses. If $j < 256$ and $T > s_j$, find $h = T - s_j$ clauses

with RANGE(c) $= j$ and ACT(c) as small as possible, and bump their range up to $j+1$. (This can be done by putting the first $m_j - h$ of them into a heap, then repeatedly bumping the least active as the remaining h are encountered; see exercise 6.1–22.)

Finally, go again through all the learned clauses c, in order of increasing c, ignoring c if RANGE(c) $> j$, otherwise copying it into a new location $c' \le c$. (Permanently false literals, which are currently defined at level 0, can also be removed at this time; thus the clause's size in MEM[$c'-1$] might be less than MEM[$c-1$]. It is possible, but unlikely, that a learned clause becomes reduced to a unit in this way, or even that it becomes empty.) The activity score in MEM[$c-5$] should be copied into MEM[$c'-5$]; but RANGE(c) and the watch links in MEM[$c-2$] and MEM[$c-3$] needn't be copied.

When copying is complete, all the watch lists should be recomputed from scratch, as in answer 260, including original clauses as well as the learned clauses that remain.

289. By induction, $y_k = (2-2^{1-k})\Delta + (2(k-2)+2^{2-k})\delta$ for all $k \ge 0$.

290. Set $k \leftarrow$ HEAP[0]; then if VAL(k) ≥ 0, delete k from the heap as in answer 262, and repeat this loop.

291. OVAL(49) will be the even number 36, because of the propagations on level 18 that led to (115).

292. If AGILITY $\ge 2^{32} - 2^{13}$, then (127) either subtracts $2^{19}-1$ or adds 1. Hence there's a minuscule chance that AGILITY will overflow by passing from $2^{32}-1$ to 2^{32} (zero). (But overflow won't be a calamity even if — unbelievably — it happens. So this is one "bug" in the author's program that he will *not* try to fix.)

293. Maintain integers $u_{\rm f}$, $v_{\rm f}$, and $\theta_{\rm f}$, where $\theta_{\rm f}$ has 64 bits. Initially $u_{\rm f} = v_{\rm f} = M_{\rm f} = 1$. When $M \ge M_{\rm f}$ is detected in step C5, do this: If $u_{\rm f} \,\&\, {-u_{\rm f}} = v_{\rm f}$, set $u_{\rm f} \leftarrow u_{\rm f}+1$, $v_{\rm f} \leftarrow 1$, $\theta_{\rm f} \leftarrow 2^{32}\psi$; otherwise set $v_{\rm f} \leftarrow 2v_{\rm f}$ and $\theta_{\rm f} \leftarrow \theta_{\rm f} + (\theta_{\rm f} \gg 4)$. Flush if AGILITY $\le \theta_{\rm f}$.

294. We have, for example, $g_{1100} = \frac{z}{3}(g_{0100}+g_{1000}+g_{1110})$, and $g_{01*1} = 1$. The solution is $g_{00*1} = g_{01*0} = g_{11*1} = A/D$, $g_{00*0} = g_{10*1} = g_{11*0} = B/D$, $g_{10*0} = C/D$, where $A = 3z - z^2 - z^3$, $B = z^2$, $C = z^3$, $D = 9 - 6z - 3z^2 + z^3$. Hence the overall generating function is $g = (6A + 6B + 2C + 2D)/(16D)$; and we find $g'(1) = 33/4$, $g''(1) = 147$. Thus mean$(g) = 8.25$, var$(g) = 87.1875$, and the standard deviation is ≈ 9.3.

295. Consider all $3\binom{n}{3}$ clauses $\bar{x}_i \vee x_j \vee x_k$ for distinct $\{i,j,k\}$, plus two additional clauses $(\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_4 \vee \bar{x}_5 \vee \bar{x}_6)$ to make the solution $0\ldots0$ unique. Only the two latter clauses cause the variables X_t and Y_t in the proof of Theorem U to deviate from each other. [C. Papadimitriou, *Computational Complexity* (1994), Problem 11.5.6. These clauses spell trouble for a lot of other SAT algorithms too.]

296. The hinted ratio $2(2p+q+1)(2p+q)/(9(p+1)(p+q+1))$ is ≈ 1 when $p \approx q$ (more precisely when $p = q - 7 + O(1/q)$). And $f(q+1,q+1)/f(q,q) = 2(n-q)(3q+3)^{\underline{3}}/(27(q+1)^2(2q+2)^{\underline{2}})$ is ≈ 1 when $q \approx n/3$. Finally, $f(n/3,n/3) = \frac{3}{4\pi n}(3/4)^n(1+O(1/n))$ by Stirling's approximation, when $n = 3q$.

297. (a) $G_q(z) = (z/3)^q C(2z^2/9)^q = G(z)^q$ where $G(z) = \bigl(3 - \sqrt{9-8z^2}\bigr)/(4z)$, by Eqs. 7.2.1.6–(18) and (24). [See *Algorithmica* **32** (2002), 620–622.]

(b) $G_q(1) = 2^{-q}$ is the probability that Y_t actually reaches 0, for some finite t.

(c) If the Y process does stop, $G_q(z)/G_q(1) = (2G(z))^q$ describes the distribution of stopping times. Hence $G_q'(1)/G_q(1) = 2qG'(1) = 3q$ is the mean length of the random walk, *given* that it terminates. (The variance, incidentally, is $24q$. A random Y-walker who doesn't finish quickly is probably doomed to wander forever.)

(d) The generating function for T, the stopping time of the Y process, is $T(z) = \sum_q \binom{n}{q} 2^{-n} G_q(z)$; and T is finite with probability $T(1) = (\frac{3}{4})^n$ by (b). If we restrict

heap
OVAL
overflow
bug
author
standard deviation
Papadimitriou
variance

consideration to such scenarios, the mean $T'(1)/T(1)$ is n; and Markov's inequality tells us that $\Pr(T \ge N) \le n/N$.

(e) The algorithm succeeds with probability $p > \Pr(T < N) \ge (1-n/N)(3/4)^n$, when it is given satisfiable clauses. So it fails after $K(4/3)^n$ trials with probability less than $\exp(K(4/3)^n \ln(1-p)) < \exp(-K(4/3)^n p) < \exp(-K/2)$ when $N = 2n$.

Markov's inequality
Papadimitriou
random number generator
biased random bits
random bits, biased

298. Change 1/3 and 2/3 in (129) to $1/k$ and $(k-1)/k$. The effect is to change $G(z)$ to $(z/k)C((k-1)z^2/k^2)$, with $G(1) = 1/(k-1)$ and $G'(1) = k/((k-1)(k-2))$. As before, $T(1) = 2^{-n}(1+G(1))^n$ and $T'(1)/T(1) = nG'(1)/(1+G(1))$. So the generalized Corollary W gives success probability $> 1 - e^{-K/2}$ when we apply Algorithm P $K(2-2/k)^n$ times with $N = \lfloor 2n/(k-2) \rfloor$.

299. In this case $G(z) = \bigl(1 - \sqrt{1-z^2}\bigr)/z$; thus $G(1) = T(1) = 1$. But $G'(1) = \infty$, so we must use a different method. The probability of failure if $N = n^2$ is

$$\frac{1}{2^n}\sum_{p,q}\binom{n}{q}\frac{q}{2p+q}\binom{2p+q}{p}\frac{[2p+q>n^2]}{2^{2p+q}} = \sum_{t>n^2}\frac{2^{-n-t}}{t}\sum_p\binom{n}{t-2p}\binom{t}{p}(t-2p)$$
$$\le \sum_{t>n^2}\frac{2^{-n-t}}{t}\binom{t}{\lfloor t/2\rfloor}\sum_p\binom{n}{t-2p}(t-2p) = \frac{n}{4}\sum_{t>n^2}\frac{2^{-t}}{t}\binom{t}{\lfloor t/2\rfloor}$$
$$< \frac{n}{4}\sum_{t>n^2}\sqrt{\frac{2}{\pi t^3}} = \frac{n}{\sqrt{8\pi}}\int_{n^2}^{\infty}\frac{dx}{\lceil x^{3/2}\rceil} < \frac{n}{\sqrt{8\pi}}\int_{n^2}^{\infty}\frac{dx}{x^{3/2}} = \frac{1}{\sqrt{2\pi}}.$$

[See C. Papadimitriou, *Computational Complexity* (1994), Theorem 11.1.]

300. In this algorithm, variables named with uppercase letters (except C and N) denote bit vectors of some fixed size (say 64); each bit position represents a separate trial. The notation U_r stands for a vector of *random* bits, each of which is 1 with probability $1/r$, independently of all other bits and all previous U's. The maximum number of flips per bit position in this variant of Algorithm P is only *approximately* equal to N.

P1′. [Initialize.] Set $X_i \leftarrow U_2$ for $1 \le i \le n$. Also set $t \leftarrow 0$.

P2′. [Begin pass.] Set $Z \leftarrow 0$ and $j \leftarrow 0$. (Flipped positions are remembered in Z.)

P3′. [Move to next clause.] If $j = m$, go to P5′. Otherwise set $j \leftarrow j+1$.

P4′. [Flip.] Let C_j be the clause $(l_1 \vee \cdots \vee l_k)$. Set $Y \leftarrow \bar{L}_1 \mathbin{\&} \cdots \mathbin{\&} \bar{L}_k$, where L_i denotes X_h if $l_i = x_h$ and L_i denotes $\bar{X}_h$ if $l_i = \bar{x}_h$. (Thus Y has 1s in positions that violate clause C_j.) Set $Z \leftarrow Z \mid Y$ and $t \leftarrow t + (Y \mathbin{\&} 1)$. Then for $r = k, k-1, \ldots, 2$ set $Y' \leftarrow Y \mathbin{\&} U_r$, $L_r \leftarrow L_r \oplus Y'$, $Y \leftarrow Y - Y'$. Finally set $L_1 \leftarrow L_1 \oplus Y$ and return to P3′.

P5′. [Done?] If $Z \ne -1$, terminate successfully: One solution is given by the bits $(X_1 \mathbin{\&} B)\ldots(X_n \mathbin{\&} B)$, where $B = \bar{Z} \mathbin{\&} (Z+1)$. Otherwise, if $t > N$, terminate unsuccessfully. Otherwise return to P2′. ▌

The shenanigans in step P4′ have the effect of flipping the offending bits of each literal with probability $1/k$, thus distributing the 1s of Y in an unbiased fashion.

301. In practice we can assume that all clauses have limited size, so that (say) $k \le 4$ in step P4′. The clauses can also be sorted by size.

A traditional random number generator produces bits U_2; and one can use $U_2 \& U_2$ to get U_4. The method of exercise 3.4.1–25 can be used for other cases; for example,

$$U_2 \mathbin{\&} (U_2 \mid (U_2 \mathbin{\&} (U_2 \mid (U_2 \mathbin{\&} (U_2 \mid (U_2 \mathbin{\&} (U_2 \mid (U_2 \mathbin{\&} U_2))))))))$$

is a sufficiently close approximation to U_3. The random numbers needed in step P1′ must be of top quality; but those used in step P4′ don't have to be especially accurate, because most of their bits are irrelevant. We can precompute the latter, making tables of 2^d values for each of U_2, U_3, U_4, and running through them cyclically by means of table indices `U2P`, `U3P`, `U4P` as in the code below, where $\texttt{UMASK} = 2^{d+3}-1$. The values of `U2P`, `U3P`, and `U4P` should be initialized to (truly) random bits whenever step P2′ starts a new pass over the clauses.

Here is sample code for the inner loop, step P4′, for clauses with $k = 3$. The octabyte in memory location $\texttt{L}+8(i-1)$ is the address in memory where X_h is stored, plus 1 if it should be complemented; for example, if l_2 is $\bar{x}_3$, the address $\texttt{X}+3\times 8+1$ will be in location $\texttt{L}+8$, where `L` is a global register. Register `mone` holds the constant -1.

`LDOU $1,L,0`	$\mathrm{addr}(L_1)$	`XOR $9,$6,$0`	$\bar{L}_3$	`STOU $6,$3,0`	$\lvert L_3\rvert \oplus Y'$
`LDOU $4,$1,0`	$\lvert L_1\rvert$	`AND $7,$7,$8`		`SUBU $7,$7,$0`	
`LDOU $2,L,8`	$\mathrm{addr}(L_2)$	`AND $7,$7,$9`	Y	`LDOU $0,U2,U2P`	
`LDOU $5,$2,0`	$\lvert L_2\rvert$	`OR Z,Z,$7`	$Z \mid Y$	`ADD U2P,U2P,8`	
`LDOU $3,L,16`	$\mathrm{addr}(L_3)$	`AND $0,$7,1`	$Y \,\&\, 1$	`AND U2P,U2P,UMASK`	
`LDOU $6,$3,0`	$\lvert L_3\rvert$	`ADD T,T,$0`	new t	`AND $0,$0,$7`	$U_2 \,\&\, Y$
`ZSEV $0,$1,mone`		`LDOU $0,U3,U3P`		`XOR $5,$5,$0`	
`XOR $7,$4,$0`	$\bar{L}_1$	`ADD U3P,U3P,8`		`STOU $5,$2,0`	$\lvert L_2\rvert \oplus Y'$
`ZSEV $0,$2,mone`		`AND U3P,U3P,UMASK`		`SUBU $7,$7,$0`	
`XOR $8,$5,$0`	$\bar{L}_2$	`AND $0,$0,$7`	$U_3 \,\&\, Y$	`XOR $4,$4,$7`	
`ZSEV $0,$3,mone`		`XOR $6,$6,$0`		`STOU $4,$1,0`	$\lvert L_1\rvert \oplus Y$ ▮

302. Assume that literals are represented internally as in Algorithm A, and that all clauses have strictly distinct literals. An efficient implementation actually requires more arrays than are stated in the text: We need to know exactly which clauses contain any given literal, just as we need to know the literals of any given clause. And we also need a (small) array $b_0 \ldots b_{k-1}$ to list the best candidate literals in step W4:

- **W4.** [Choose l.] Set $c \leftarrow \infty$, $j \leftarrow 0$, and do the following while $j < k$: Set $j \leftarrow j+1$, $l \leftarrow l_j$; and if $c_{|l|} < c$, set $c \leftarrow c_{|l|}$, $b_0 \leftarrow l$, $s \leftarrow 1$; or if $c_{|l|} = c$, set $b_s \leftarrow l$, $s \leftarrow s+1$. Then if $c = 0$, or if $c \ge 1$ and $U \ge p$, set $l \leftarrow b_{\lfloor sU \rfloor}$; otherwise set $l \leftarrow l_{\lfloor kU \rfloor + 1}$. (Each random fraction U is independent of the others.)
- **W5.** [Flip l.] Set $s \leftarrow 0$. For each j such that C_j contains l, make clause C_j happier as follows: Set $q \leftarrow k_j$, $k_j \leftarrow q+1$; and if $q = 0$, set $s \leftarrow s+1$ and delete C_j from the f array (see below); or if $q = 1$, decrease the cost of C_j's critical variable (see below). Then set $c_{|l|} \leftarrow s$ and $x_{|l|} \leftarrow \bar{x}_{|l|}$. For each j such that C_j contains $\bar{l}$, make clause C_j sadder as follows: Set $q \leftarrow k_j - 1$, $k_j \leftarrow q$; and if $q = 0$, insert C_j into the f array (see below); or if $q = 1$, increase the cost of C_j's critical variable (see below). Set $t \leftarrow t+1$ and return to W2. ▮

To insert C_j into f, we set $f_r \leftarrow j$, $w_j \leftarrow r$, and $r \leftarrow r+1$ (as in step W1). To delete it, we set $h \leftarrow w_j$, $r \leftarrow r-1$, $f_h \leftarrow f_r$, $w_{f_r} \leftarrow h$.

Whenever we want to update the cost of C_j's critical variable in step W5, we know that C_j has exactly one true literal. Thus, if the literals of C_j appear sequentially in a master array M, it's easy to locate the critical variable $x_{|M_i|}$: We simply set $i \leftarrow \texttt{START}(j)$; then while M_i is false (namely while $x_{|M_i|} = M_i \,\&\, 1$), set $i \leftarrow i+1$.

A slight refinement is advantageous when we will be increasing $c_{|M_i|}$: If $i \ne \texttt{START}(j)$, swap $M_{\texttt{START}(j)} \leftrightarrow M_i$. This change significantly shortens the search when $c_{|M_i|}$ is subsequently decreased. (In fact, it reduced the total running time by more than 5% in the author's experiments with random 3SAT problems.)

branchless computation
ZSEV
`mone`
swapping to the front
Knuth

303. In this case $D = 3 - z - z^2 = A/z$, and we have $g'(1) = 3$, $g''(1) = 73/4$. Thus $\text{mean}(g) = 3$ and $\text{var}(g) = 12.25 = 3.5^2$. random walk

304. If $\nu x = x_1 + \cdots + x_n = a$, there are $a(n-a)$ unsatisfied clauses; hence there are two solutions, $0\ldots0$ and $1\ldots1$. If $x_1 \ldots x_n$ isn't a solution, Algorithm P will change a to $a \pm 1$, each with probability $\frac{1}{2}$. Thus the probability generating function g_a for future flips is 1 when $a = 0$ or $a = n$, otherwise it is $z(g_{a-1} + g_{a+1})/2$. And the overall generating function is $g = \sum_a \binom{n}{a} g_a/2^n$. Clearly $g_a = g_{n-a}$.

Exercise MPR–105 determines g_a and proves that the mean number of flips, $g'_a(1)$, is $a(n-a)$ for $0 \le a \le n$. Thus $g'(1) = 2^{-n}\sum_{a=0}^{n}\binom{n}{a} g'_a(1) = \frac{1}{2}\binom{n}{2}$.

Turning now to Algorithm W, again with $x_1 + \cdots + x_n = a$, the cost of x_i is $a-1$ when $x_i = 1$, $n-a-1$ when $x_i = 0$. Therefore $g_1 = g_{n-1} = z$ in this case. And for $2 \le a \le n-2$, we will move closer to a solution with probability q and farther from a solution with probability p, where $p + q = 1$ and $p = p'/2 \le 1/2$; here p' is the greed-avoidance parameter of Algorithm W. Thus for $2 \le a \le n/2$ we have $g_a = g_{n-a} = z(qg_{a-1} + pg_{a+1})$.

If $p' = 0$, so that the walk is 100% greedy, Algorithm W zooms in on the solution, with $g_a = z^a$. Exercise 1.2.6–68 with $p = 1/2$ tells us that $g'(1) = n/2 - m\binom{n}{m}/2^n = n/2 - \sqrt{n/2\pi} + O(1)$ in that case. On the other hand if $p' = 1$, so that the walk is greedy only when $a = 1$ or $a = n-1$, we're almost in the situation of Algorithm P but with n decreased by 2. Then $g'(1) = 2^{-n}\sum_{a=1}^{n-1}\binom{n}{a}(1 + (a-1)(n-2) - (a-1)^2) = n(n-5)/4 + 2 + (2n-4)/2^n$; greed triumphs.

What happens as p' rises from 0 to 1? Let's decrease n by 2 and use the rule $g_a = z(qg_{a-1} + pg_{a+1})$ for $1 \le a \le n/2$, so that the calculations are similar to those we did for Algorithm P but with p now $\le 1/2$ instead of $p = 1/2$. Functions t_m and u_m can be defined as above; but $g_a = (qz)^a t_{m-a}/t_m$, the new recurrence is $t_{m+1} = t_m - pqz^2 t_{m-1}$, and $t_0 = 1/q$, $u_0 = 1/(qz)$. These functions are polynomials in p, q, and z, whose coefficients are binomial coefficients: In the notation of exercise 1.2.9–15, for $m > 0$ we have $t_m = G_{m-1}(-pqz^2) - pz^2 G_{m-2}(-pqz^2)$ and $u_m = G_{m-1}(-pqz^2) - pzG_{m-2}(-pqz^2)$, so

$$T(w) = \frac{1 - pw}{q(1 - w + pqz^2w^2)}; \qquad U(w) = \frac{1 - (1 - qz)w}{qz(1 - w + pqz^2w^2)}.$$

Consequently $t'_m(1)/t_m(1) = 2pq(1-(p/q)^m)/(q-p)^2 - 2pm/(q-p)$ and $u'_m(1)/u_m(1) = (2p - (p/q)^m q)/(q-p)^2 - 2p(m - \frac{1}{2})/(q-p)$; $g'_a(1) = a/(q-p) - 2pq((q/p)^a - 1)/(q-p)^2$ for $0 \le a \le n/2$ when n is even, $a/(q-p) - q((q/p)^a - 1)/(q-p)^2$ when n is odd. The overall totals when $n = 1000$ and $p' = (.001, .01, .1, .5, .9, .99, .999)$ are respectively $\approx (487.9, 492.3, 541.4, 973.7, 4853.4, 44688.2, 183063.4)$.

305. That little additional clause reverses the picture! Now there's only one solution, and greediness fails badly when $\nu x > n/2$ because it keeps trying to move x *away* from the solution. To analyze the new situation in detail, we need $3(n-1)$ generating functions g_{ab}, where $a = x_1 + x_2$ and $b = x_3 + \cdots + x_n$. The expected number of flips will be $g'(1)$, where $g = 2^{-n}\sum_{a=0}^{2}\sum_{b=0}^{n-2}\binom{2}{a}\binom{n-2}{b} g_{ab}$.

The behavior of Algorithm P is ambiguous, because the unsatisfied clause found in step P2 depends on the clause ordering. The most favorable case arises when $a = 2$, because we can decrease a to 1 by working on the special clause $\bar{x}_1 \vee \bar{x}_2$. Any other clause is equally likely to increase or decrease $a + b$. So the best-case generating functions maximize the chance of reaching $a = 2$: $g_{00} = 1$, $g_{01} = \frac{z}{2}(g_{00} + g_{11})$, $g_{02} = \frac{z}{2}(g_{01} + g_{12})$, $g_{10} = \frac{z}{2}(g_{00} + g_{20})$, $g_{11} = \frac{z}{2}(g_{10} + g_{21})$, $g_{12} = \frac{z}{2}(g_{11} + g_{22})$, and $g_{2b} = zg_{1b}$. The solution has $g_{1b} = \bigl(z/(2 - z^2)\bigr)^{b+1}$; and we find $\text{mean}(g) = 183/32 = 5.71875$.

infinite loop
geometric distribution
memoryless

The worst case arises whenever $g_{20} \neq zg_{10}$ and $g_{21} \neq zg_{11}$; for example we can take $g_{20} = \frac{z}{2}(g_{10}+g_{21})$, $g_{21} = \frac{z}{2}(g_{20}+g_{22})$, together with the other seven equations from the best case. Then $g_{01} = g_{10} = z(4-3z^2)/d$, $g_{02} = g_{11} = g_{20} = z^2(2-z^2)/d$, and $g_{12} = g_{21} = z^3/d$, where $d = 8-8z^2+z^4$. Overall, $g = (1+z)^2(2-z^2)/(4d)$ and $\mathrm{mean}(g) = 11$.

(This analysis can be extended to larger n: The worst case turns out to have $g_{ab} = g_{a+b} = (z/2)^{a+b}t_{n-a-b}/t_n$, in the notation of the previous exercise, giving $n(3n-1)/4$ flips on average. The best case has g_{1b} as before; hence $g'_{0b} = 3b+2-2^{1-b}$, $g'_{1b} = 3b+3$, and $g'_{2b} = 3b+4$ when $z = 1$. The best average number of flips therefore turns out to be *linear*, with $\mathrm{mean}(g) = \frac{3}{2}n - \frac{8}{9}(3/4)^n$.)

The analysis becomes more exciting, but trickier, when we use Algorithm W. Let $p = p'/2$ and $q = 1-p$ as in the previous answer. Clearly $g_{00} = 1$, $g_{01} = g_{10} = zg_{00}$, $g_{02} = \frac{z}{2}(g_{01}+g_{12})$, and $g_{22} = zg_{12}$; but the other four cases need some thought. We have

$$g_{11} \;=\; \frac{z}{4}\big((\tfrac{1}{2}+q)(g_{01}+g_{10}) + g_{12} + 2pg_{21}\big),$$

since the costs for $x_1x_2x_3x_4 = 1010$ are 1211 and the unsatisfied clauses are $(\bar{x}_1 \vee x_4)$, $(\bar{x}_3 \vee x_4)$, $(\bar{x}_1 \vee x_2)$, $(\bar{x}_3 \vee x_2)$; in the former two clauses we flip each literal equally often, but in the latter two we flip x_2 with probability p and the other with probability q. A similar but simpler analysis shows that $g_{21} = \frac{z}{4}(g_{11}+3g_{22})$ and $g_{20} = \frac{z}{5}(3g_{10}+2g_{21})$.

The most interesting case is $g_{12} = \frac{z}{3}(pg_{02} + 2pg_{11} + 3qg_{22})$, where the costs are 2122 and the problematic clauses are $(\bar{x}_1 \vee x_2)$, $(\bar{x}_3 \vee x_2)$, $(\bar{x}_4 \vee x_2)$. If $p = 0$, Algorithm W will always decide to flip x_2; but then we'll be back in state 12 after the next flip.

Indeed, setting $p = 0$ yields $g_{00} = 1$, $g_{01} = g_{10} = z$, $g_{02} = \frac{1}{2}z^2$, $g_{11} = \frac{3}{4}z^2$, $g_{20} = \frac{3}{5}z^2 + \frac{3}{40}z^4$, $g_{21} = \frac{3}{16}z^3$, and $g_{12} = g_{22} = 0$. The weighted total therefore turns out to be $g = (40 + 160z + 164z^2 + 15z^3 + 3z^4)/640$. Notice that the greedy random walk never succeeds after making more than 4 flips, in this case; so we should set $N = 4$ and restart after each failure. The probability of success is $g(1) = 191/320$. (This strategy is actually quite good: It succeeds after making an average of $1577/382 \approx 4.13$ flips and choosing random starting values $x_1x_2x_3x_4$ about 320/191 times.)

If p is positive, no matter how tiny, the success probability for $N = \infty$ is $g(1) = 1$. But the denominator of g is $48 - 48z^2 + 26pz^2 + 6pz^4 - 17p^2z^4$, and we find that $\mathrm{mean}(g) = (1548+2399p-255p^2)/(1280p-680p^2) = (6192+4798p'-255p'^2)/(2560p'-680p'^2)$. Taking $p' = (.001, .01, .1, .5, .9, .99, .999)$ in this formula gives, respectively, the approximate values (2421.3, 244.4, 26.8, 7.7, 5.9, 5.7, 5.7).

(Calculations for $n = 12$ show that g is a polynomial of degree 8 when $p = 0$, with $g(1) \approx .51$ and $g'(1) \approx 2.40$. Thus, setting $N = 8$ yields success after about 16.1 flips and 1.95 initializations. When $p > 0$ we have $g'(1) \approx 1.635p^{-5} + O(p^{-4})$ as $p \to 0$, and the seven values of p' considered above yield respectively (5×10^{16}, 5×10^{11}, 5×10^6, 1034.3, 91.1, 83.89, 83.95) flips — surprisingly *not* monotone decreasing in p'. These WalkSAT statistics can be compared with 17.97 to 105 flips for Algorithm P.)

306. (a) Since $l(N) = E_N + (1-q_N)(N + l(N))$, we have $q_N l(N) = E_N + N - Nq_N = p_1 + 2p_2 + \cdots + Np_N + Np_{N+1} + \cdots + Np_\infty = N - (q_1 + \cdots + q_{N-1})$.

(b) If $N = m+k$ and $k \geq 0$ we have $E_N = m^2/n$, $q_1 + \cdots + q_{N-1} = km/n$, and $q_N = m/n$; hence $l(N) = n + k(n-m)/m$.

(c) If $N \leq n$, $l(N) = \big(N - \binom{N}{2}/n\big)/(N/n) = n - \frac{N-1}{2}$; otherwise $l(N) = l(n) = \frac{n+1}{2}$.

(d) From $q_N = p_1(N - q_1 - \cdots - q_{N-1})$ and $q_{N+1} = p_1(N + 1 - q_1 - \cdots - q_N)$ we deduce $p_{N+1} = p_1(1-q_N)$, hence $1 - q_{N+1} = (1-p_1)(1-q_N)$. So it's a geometric distribution, with $p_t = p(1-p)^{t-1}$ for $t \geq 1$. (The fact that $l(1) = l(2) = \cdots$ is called the "memoryless property" of the geometric distribution.)

(e) Choose $p_1, \ldots, p_n$ arbitrarily, with $q_n = p_1 + \cdots + p_n \le 1$. Then, arguing as in (d), $p_{n+1}, p_{n+2}, \ldots$ are defined by $1 - q_N = (1 - 1/l(n))^{N-n}(1 - q_n)$ for $N \ge n$.

(f) Since $l(n+1) - l(n) = (n - (q_1 + \cdots + q_n))(1 - 1/q_n) \le 0$, we must have $q_n = 1$ and $l(n) = l(n+1)$. (The case $l(n) < l(n+1)$ is impossible.)

(g) Let $x = p_1$ and $y = p_2$. By part (f), the conditions are equivalent to $0 < x \le x+y < 1$ and $x(3-2x-y) > 1$. Hence $0 < (2x-1)(1-x) - xy \le (2x-1)(1-x)$; we get the general solution by first choosing $\frac{1}{2} < x < 1$, then $0 \le y < (2x-1)(1-x)/x$.

(h) If $N^* = \infty$ and $l(n) < \infty$, we can find n' with $q_{n'} l(n') = p_1 + 2p_2 + \cdots + n'p_{n'} + n'p_{n'+1} + \cdots + n'p_\infty > l(n)$. Hence $l(N) \ge q_N l(N) \ge q_{n'} l(n') > l(n)$ for all $N \ge n'$.

(i) We have $q_{n+k} = k/(k+1)$ for $k \ge 0$; hence $l(n+k) = (k+1)(n+H_k)/k$. The minimum occurs when $l(n+k) \approx l(n+k-1)$, namely when $n \approx k - H_k$; thus $k = n + \ln n + O(1)$. For example, the optimum cutoff value when $n = 10$ is $N^* = 23$. (Notice that $E_\infty = \infty$, yet $l = l(N^*) \approx 14.194$ in this case.)

(j) Let $p_t = [t > 1]/2^{t-1}$. Then $l(N) = (3 - 2^{2-N})/(1 - 2^{1-N})$ decreases to 3.

(k) Clearly $l \le L$. For $N \le L$ we have $l(N) = (N - (q_1 + \cdots + q_{N-1}))/q_N \ge (N - (1 + \cdots + (N-1))/L)/(N/L) = L - (N-1)/2 \ge (L+1)/2$. And for $N = \lfloor L \rfloor + k + 1$, similarly, $l(N) \ge N - (1 + \cdots + \lfloor L \rfloor + kL)/L = \lfloor L+1 \rfloor (1 - \lfloor L \rfloor/(2L)) \ge (L+1)/2$.

307. (a) $\mathrm{E}\, X = E_{N_1} + (1 - q_{N_1})(N_1 + \mathrm{E}\, X')$, where X' is the number of steps for the sequence $(N_2, N_3, \ldots)$. For numerical results, start with $j \leftarrow 0$, $s \leftarrow 0$, $\alpha \leftarrow 1$; then, while $\alpha > \epsilon$, set $j \leftarrow j+1$, $\alpha \leftarrow (1 - q_{N_j})\alpha$, and $s \leftarrow s + E_{N_j} + \alpha N_j$. (Here ϵ is tiny.)

(b) Let $P_j = (1 - q_{N_1}) \ldots (1 - q_{N_{j-1}}) = \Pr(X > T_j)$, and note that $P_j \le (1 - p_n)^{j-1}$ where $n = \min\{t \mid p_t > 0\}$. Since $q_N l(N) = E_N + (1 - q_N)N$, we have

$$\begin{aligned} \mathrm{E}\, X &= q_{N_1} l(N_1) + (1 - q_{N_1})\bigl(q_{N_2} l(N_2) + (1 - q_{N_2})\bigl(q_{N_3} l(N_3) + \cdots\bigr)\bigr) \\ &= \sum_{j=1}^{\infty} P_j q_{N_j} l(N_j) = \sum_{j=1}^{\infty} (P_j - P_{j+1})\, l(N_j). \end{aligned}$$

(c) $\mathrm{E}\, X \ge \sum_{j=1}^{\infty} (P_j - P_{j+1}) l(N^*) = l$.

(d) We can assume that $N_j \le n$ for all j; otherwise the strategy would do even worse. For the hint, let $\{N_1, \ldots, N_r\}$ contain r_m occurrences of m, for $1 \le m \le n$, and suppose $t_m = r_m + \cdots + r_n$. If $t_m < n/(2m)$, the probability of failure would be $(1 - m/n)^{t_m} \ge 1 - t_m m/n > 1/2$. Hence we have $t_m \ge n/(2m)$ for all m, and $N_1 + \cdots + N_r = t_1 + \cdots + t_n \ge nH_n/2$.

Now there's some m such that the first $r-1$ trials fail on $p^{(m)}$ with probability $> \frac{1}{2}$. For this m we have $\mathrm{E}\, X > \frac{1}{2}(N_1 + \cdots + N_{r-1}) \ge \frac{1}{2}(N_1 + \cdots + N_r - n)$.

308. (a) $2^{a+1} - 1$; and we also have $S_{2^a + b} = S_{b+1}$ for $0 \le b < 2^a - 1$ (by induction).

(b) The sequence (u_n, v_n) in (131) has $1 + \rho k$ entries with $u_n = k$; and $\rho 1 + \cdots + \rho n = n - \nu n$ by Eq. 7.1.3–(61). From the double generating function $g(w,z) = \sum_{n \ge 0} w^{\nu n} z^n = (1+wz)(1+wz^2)(1+wz^4)(1+wz^8) \ldots$ we deduce that $\sum_{k \ge 0} z^{2k+1-\nu k} = zg(z^{-1}, z^2)$.

(c) $\{n \mid S_n = 2^a\} = \{2^{a+1}k + 2^{a+1} - 1 - \nu k \mid k \ge 0\}$; hence $\sum_{n \ge 0} z^n [S_n = 2^a] = z^{2^{a+1}-1} g(z^{-1}, z^{2^{a+1}}) = z^{2^{a+1}-1}(1 + z^{2^{a+1}-1})(1 + z^{2^{a+2}-1})(1 + z^{2^{a+3}-1}) \ldots$.

(d) When 2^a occurs for the 2^bth time, we've had $2^{a+b-c} - [c > a]$ occurrences of 2^c, for $0 \le c \le a+b$. Consequently $\Sigma(a,b,1) = (a+b-1)2^{a+b} + 2^{a+1}$.

(e) The exact value is $\sum_{c=0}^{a+b} 2^{a+b-c} 2^c + \sum_{c=1}^{\rho k} 2^{a+b+c}$; and $\rho k \le \lambda k = \lfloor \lg k \rfloor$.

(f) The stated formula is $\mathrm{E} \min_k \{\Sigma(a,b,k) \mid \Sigma(a,b,k) \ge X\}$, if we penalize the algorithm so that it *never* succeeds unless it is run with the particular cutoff $N = 2^a$.

flush
Fibonacci ruler function
ruler of Fibonaccis
Cohen

(g) We have $Q \le (1-q_t)^{2^b} \le (1-q_t)^{1/q_t} < e^{-1}$; hence $\mathrm{E}\,X < (a+b-1)2^{a+b} + 2^{a+1} + \sum_{k=1}^{\infty}(a+b+2k-1)2^{a+b}e^{-k} = 2^{a+b}\bigl((a+b)e/(e-1)+e(3-e)/(e-1)^2+2^{1-b}\bigr)$. Furthermore we have $2^{a+b} < 8l - 4l[b=0]$, by exercise 306(k).

309. No — far from it. If Algorithm C were to satisfy the hypotheses of exercise 306, it would have to do *complete* restarts: It would not only have to flush *every* literal from the trail, it would also have to forget all the clauses that it has learned, and reinitialize the random heap. [But reluctant doubling appears to work well also outside of Vegas.]

310. A method analogous to (131) can be used: Let $(u'_1, v'_1) = (1, 0)$; then define $(u'_{n+1}, v'_{n+1}) = \bigl(u'_n \mathbin{\&} -u'_n = 1 \ll v'_n?\ (\mathrm{succ}(u'_n), 0)\colon (u'_n, v'_n + 1)\bigr)$. Here 'succ' is the Fibonacci-code successor function that is defined by six bitwise operations in answer 7.1.3–158. Finally, let $S'_n = F_{v'_n+2}$ for $n \ge 1$. (This sequence $\langle S'_n\rangle$, like $\langle S_n\rangle$, is "nicely balanced"; hence it is universal as in exercise 308. For example, when F_a appears for the first time, there have been exactly F_{a+2-c} occurrences of F_c, for $2 \le c \le a$.)

311. Because $\langle R_n\rangle$ does surprisingly well in these tests, it seems desirable to consider also its Fibonacci analog: If $f_n = \mathrm{succ}(f_{n-1})$ is the binary Fibonacci code for n, we can call $\langle \rho' n\rangle = \langle \rho f_n\rangle = (0,1,2,0,3,0,1,4,0,\ldots)$ the "Fibonacci ruler function," and let $\langle R'_n\rangle = (1,2,3,1,5,1,2,8,1,\ldots)$ be the "ruler of Fibonaccis," where $R'_n = F_{2+\rho' n}$.

The results $(E_S, E_{S'}, E_R, E_{R'})$ for $m = 1$ and $m = 2$ are respectively $(315.1, 357.8, 405.8, 502.5)$ and $(322.8, 284.1, 404.9, 390.0)$; thus S beats S' beats R beats R' when $m = 1$, while S' beats S beats R' beats R when $m = 2$. The situation is, however, reversed for larger values of m: R beats R' beats S beats S' when $m = 90$, while R' beats R beats S' beats S when $m = 89$.

In general, the reluctant methods shine for small m, while the more "aggressive" ruler methods forge ahead as m grows: When $n = 100$, S beats R if and only if $m \le 13$, and S' beats R' if and only if $m \le 12$. The doubling methods are best when m is a power of 2 or slightly less; the Fibonacci methods are best when m is a Fibonacci number or slightly less. The worst cases occur at $m = 65 = 2^6 + 1$ for S and R (namely 1402.2 and 845.0); they occur at $m = 90 = F_{11} + 1$ for S' and R' (namely 1884.8 and 805.9).

312. $T(m,n) = m + b2^b h_0(\theta)/\theta + 2^b g(\theta)$, where $b = \lceil \lg m\rceil$, $\theta = 1 - m/n$, $h_a(z) = \sum_n z^n [S_n = 2^a]$, and $g(z) = \sum_{n\ge 1} S_n z^n = \sum_{a\ge 0} 2^a h_a(z)$.

313. If l is flipped, the number of unsatisfied clauses increases by the cost of $|l|$ and decreases by the number of unsatisfied clauses that contain l; and the latter is at least 1.

Consider the following interesting clauses, which have the unique solution 0000:

$$x_1 \vee \bar x_2,\ \ \bar x_1 \vee x_2,\ \ x_2 \vee \bar x_3,\ \ \bar x_2 \vee x_3,\ \ x_3 \vee \bar x_4,\ \ \bar x_3 \vee x_4,\ \ \bar x_1 \vee \bar x_4.$$

"Uphill" moves $1011 \mapsto 1111$ and $1101 \mapsto 1111$ are forced; also $0110 \mapsto 1110$ or 0111.

314. (Solution by Bram Cohen, 2012.) Consider the 10 clauses $\bar1\bar234\bar567$, $\bar12\bar34\bar567$, $123\bar45$, $123\bar46$, $123\bar47$, $\bar1\bar2\bar3\bar4$, $\bar1\bar2\bar3\bar5$, $\bar1\bar2\bar3\bar6$, $\bar1\bar2\bar4\bar5$, $\bar1\bar2\bar4\bar6$, and 60 more obtained by the cyclic permutation (1234567). All binary $x = x_1 \ldots x_7$ with weight $\nu x = 2$ have cost-free flips leading to weight 3, but no such flips to weight 1. Since the only solution has weight 0, Algorithm W loops forever whenever $\nu x > 1$. (Is there a smaller example?)

315. Any value with $0 \le p < 1/2$ works, since each graph component is either K_1 or K_2.

316. No; $\max \theta(1-\theta)^d$ for $0 \le \theta < 1$ is $d^d/(d+1)^{d+1}$, when $\theta = 1/(d+1)$. [But Theorem J for $d > 2$ *is* a consequence of the improved Theorem L in exercise 356(c).]

317. Number the vertices so that the neighbors of vertex 1 are 2, ..., d', and let $G_j = G \setminus \{1,\ldots,j\}$. Then $\alpha(G) = \alpha(G_1) - \Pr(A_1 \cap \overline{A}_2 \cap \cdots \cap \overline{A}_m)$, and the latter probability is $\le \Pr(A_1 \cap \overline{A}_{d'+1} \cap \cdots \cap \overline{A}_m) = \Pr(A_1 \mid \overline{A}_{d'+1} \cap \cdots \cap \overline{A}_m)\alpha(G_{d'}) \le p\alpha(G_{d'})$.

Möbius polynomial
Chebyshev polynomials
convex hull

Let $\rho = (d-1)/d$. By induction we have $\alpha(G_j) > \rho\alpha(G_{j+1})$ for $1 \le j < d'$, because vertex $j+1$ has degree $< d$ in G_j. If $d' = 1$ then $\alpha(G) \ge \alpha(G_1) - p\alpha(G_1) > \rho\alpha(G_1) > 0$. Otherwise if $d' \le d$, $\alpha(G) \ge \alpha(G_1) - p\alpha(G_{d'}) > \alpha(G_1) - p\rho^{1-d'}\alpha(G_1) \ge \alpha(G_1) - p\rho^{1-d}\alpha(G_1) = \rho\alpha(G_1) > 0$. Otherwise we must have $d' = d+1$, with vertex 1 of degree d, and $\alpha(G) > \alpha(G_1) - p\rho^{-d}\alpha(G_1) = \frac{d-2}{d-1}\alpha(G_1) \ge 0$.

318. Let $f_n = M_G(p)$ where G is the graph of a complete t-ary tree with t^n leaves; thus G has t^k vertices at distance k from the root, for $0 \le k \le n$. Then

$$f_0 = 1-p, \quad f_1 = (1-p)^t - p, \quad \text{and} \quad f_{n+1} = f_n^t - p f_{n-1}^{t^2} \text{ for } n > 1.$$

By Theorem S, it suffices to show that $f_n \le 0$ for some n.

The key idea is to let $g_0 = 1-p$ and $g_{n+1} = f_{n+1}/f_n^t = 1 - p/g_n^t$. Assuming that $g_n > 0$ for all n, we have $g_1 < g_0$ and $g_n - g_{n+1} = p/g_n^t - p/g_{n+1}^t > 0$ when $g_{n+1} < g_n$. Hence $\lim_{n\to\infty} g_n = \lambda$ exists, with $0 < \lambda < 1$. Furthermore $\lambda = 1 - p/\lambda^t$, so that $p = \lambda^t(1-\lambda)$. But then $p \le t^t/(t+1)^{t+1}$ (see answer 316 with $\theta = 1-\lambda$).

[One must admit, however, that the limit is not often reached until n is extremely large. For example, even if $t = 2$ and $p = .149$, we don't have $f_n < 0$ until $n = 45$. Thus G must have at least 2^{45} vertices before this value of p is too large for Lemma L.]

319. Let $x = 1/(d-1)$. Since $e^x > 1 + x = d/(d-1)$, we have $e > (d/(d-1))^{d-1}$.

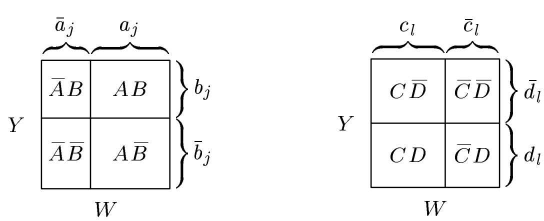
320. (a) Let $f_m(p)$ be the Möbius polynomial when $p_1 = \cdots = p_m = p$. Then we have $f_m(p) = f_{m-1}(p) - pf_{m-2}(p)$, and one can show by induction that $f_m(1/(4\cos^2\theta)) = \sin((m+2)\theta)/((2\cos\theta)^{m+1}\sin\theta)$. The threshold decreases to $1/4$ as $m \to \infty$.

(b) $1/(4\cos^2\frac{\pi}{2m})$; the Möbius polynomial $g_m(p) = f_{m-1}(p) - pf_{m-3}(p)$ satisfies the same recurrence as $f_m(p)$, and equals $2\cos m\theta/(2\cos\theta)^m$ when $p = 1/(4\cos^2\theta)$.

[In terms of the classical Chebyshev polynomials, $g_m(p) = 2p^{m/2}T_m(1/(2\sqrt{p}))$ and $f_m(p) = p^{(m+1)/2}U_{m+1}(1/(2\sqrt{p}))$.]

321. Let $\theta = (2-\sqrt{2})/2$, $\theta' = \theta(1-\theta) = (\sqrt{2}-1)/2$, and $c = (p-\theta)/(1-\theta)$. The method of answer 345 gives $(\Pr(\overline{A}\overline{B}\overline{C}\overline{D}), \Pr(A\overline{B}\overline{C}\overline{D}), \Pr(AB\overline{C}\overline{D}), \Pr(A\overline{B}C\overline{D}), \Pr(ABC\overline{D}), \Pr(ABCD)) = (0, \theta'(1-c)^3, 2\theta'(1-c)^2c, \theta^2(1-c)^2+2\theta'(1-c)^3, \theta^2(1-c)c + 3\theta'(1-c)c^2, \theta^2c^2+4\theta'c^3)$. Other cases are symmetric to these six. When $p = 3/10$ the six probabilities are $\approx (0, .20092, .00408, .08815, .00092, .00002)$.

322. (a) Let $a_j = \sum_i w_i[ij \in A]$, $b_j = \sum_k y_k[jk \in B]$, $c_l = \sum_k y_k[kl \in C]$, and $d_l = \sum_i w_i[li \in D]$. Then when $X = j$ and $Z = l$, the best way to allocate the events is



within W and Y. Hence $\Pr(\overline{A}\cap\overline{B}\cap\overline{C}\cap\overline{D}) = \sum_{j,l} x_j z_l((\bar{a}_j+\bar{d}_l)\dot{-}1)((\bar{b}_j+\bar{c}_l)\dot{-}1)$, which is zero if and only if we have $a_j + d_l \ge 1$ or $b_j + c_l \ge 1$ for all j and l with $x_j z_l > 0$.

(b) Since $\sum_j x_j(a_j, b_j) = (p, p)$, the point (p, p) lies in the convex hull of the points (a_j, b_j). So there must be points $(a, b) = (a_j, b_j)$ and $(a', b') = (a_{j'}, b_{j'})$ such that the line from (a, b) to (a', b') intersects the region $\{(x, y) \mid 0 \le x, y \le p\}$; in other words $\mu a + (1-\mu)a' \le p$ and $\mu b + (1-\mu)b' \le p$. Similarly we can find c, d, c', d', ν.

(c) Fact: If $a \ge \frac{2}{3}$ and $b' \ge \frac{2}{3}$, then $\mu = \frac{1}{2}$; hence $a = b' = \frac{2}{3}$ and $a' = b = 0$. Notice also that there are 16 symmetries, generated by (i) $a \leftrightarrow b$, $c \leftrightarrow d$; (ii) $a \leftrightarrow a'$, $b \leftrightarrow b'$, $\mu \leftrightarrow 1-\mu$; (iii) $c \leftrightarrow c'$, $d \leftrightarrow d'$, $\nu \leftrightarrow 1-\nu$; (iv) $a \leftrightarrow d$, $b \leftrightarrow c$, $\mu \leftrightarrow \nu$.

If $c \le c'$ and $d \le d'$, or if $c \le \frac{1}{3}$ and $d \le \frac{1}{3}$, we can assume (by symmetry) that the Fact applies; this gives a solution to all the constraints, with $c = d = c' = d' = \frac{1}{3}$.

For the remaining solutions we may assume that $a, b' > \frac{1}{3} > a', b$. Suppose the line from (a,b) to (a',b') intersects the line from $(0,0)$ to $(1,1)$ at the point (α,α); dividing a, b, a', b' by 3α gives a solution in which $\mu a + (1-\mu)a' = \mu b + (1-\mu)b' = \frac{1}{3}$. Similarly, we can assume that $d, c' > \frac{1}{3} > d', c$ and that $\nu c + (1-\nu)c' = \nu d + (1-\nu)d' = \frac{1}{3}$. Consequently $a + d \ge 1$ and $b' + c' \ge 1$. Symmetry also allows us to assume that $a + d' \ge 1$. In particular, $a > \frac{2}{3}$; and, by the Fact, $b' < \frac{2}{3}$. So $a' + d \ge 1$, $d > \frac{2}{3}$, $c' < \frac{2}{3}$.

Now extend the lines that connect (a,b) to (a',b') and (c,d) to (c',d'), by increasing a, b', c', d while decreasing a', b, c, d', until $a' = 1-d$ and $a = 1-d'$, and until either $a = 1$ or $b = 0$, and either $d = 1$ or $c = 0$. The only solution of this kind with $b' + c' \ge 1$ occurs when $a = d = 1$, $a' = b = c = d' = 0$, $b' = c' = 1/2$, $\mu = \frac{1}{3}$, $\nu = \frac{2}{3}$.

(d) For the first solution, we can let W, X, Y, Z be uniform on $\{0,1,2\}$, $\{0,1\}$, $\{0,1,2\}$, and $\{0\}$, respectively; and let $A = \{10,20\}$, $B = \{11,12\}$, $C = \{00\}$, $D = \{00\}$. (For example, $WXYZ = 1110$ gives event B.) The second solution turns out to be the same, but with (X,Y,Z,W) in place of (W,X,Y,Z). Notice that the solution applies also to P_4, where the threshold *is* $\frac{1}{3}$. [See *STOC* **43** (2011), 242.]

323. *cbc*. In this simple case, we just eliminate all strings in which c is followed by a.

324. For $1 \le j \le n$, and for each v such that $v = x_j$ or $v \frac{\quad}{} x_j$, let $i \prec j$ for each $i < j$ such that $v = x_i$. (If several values of i qualify, it suffices to consider only the largest one. Several authors have used the term "dependence graph" for this partial ordering.) The traces equivalent to α correspond to the topological sortings with respect to $\prec$, because those arrangements of the letters are precisely the permutations that preserve the empilement.

In (136), for example, with $x_1 \ldots x_n = bcebafdc$, we have $1 \prec 2$, $1 \prec 4$, $2 \prec 4$, $4 \prec 5$, $3 \prec 6$, $2 \prec 7$, $3 \prec 7$, $2 \prec 8$, $4 \prec 8$, and $7 \prec 8$. Algorithm 7.2.1.2V produces 105 solutions, 12345678 (*bcebafdc*) through 36127485 (*efbcdbca*).

325. Every such trace α yields an acyclic orientation, if we let $u \longrightarrow v$ when u appears at a lower level in α's empilement. Conversely, the topological sortings of any acyclic orientation are all equivalent traces; so this correspondence is one-to-one. [See Ira M. Gessel, *Discrete Mathematics* **232** (2001), 119–130.]

326. True: x commutes with y if and only if y commutes with x.

327. Each trace α is represented by its height $h = h(\alpha) \ge 0$, and by h linked lists $L_j = L_j(\alpha)$ for $0 \le j < h$. The elements of L_j are the letters on level j of α's empilement; these letters have disjoint territories, and we keep each list in alphabetic order so that the representation is unique. The canonical string representing α is then $L_0 L_1 \ldots L_{h-1}$. (For example, in (136) we have $L_0 = be$, $L_1 = cf$, $L_2 = bd$, $L_3 = ac$, and the canonical representation is *becfbdac*.) We also maintain the sets $U_j = \bigcup\{T(a) \mid a \in L_j\}$ as bit vectors; in (136), for example, they are $U_0 = {}^\#\mathtt{36}$, $U_1 = {}^\#\mathtt{1b}$, $U_2 = {}^\#\mathtt{3c}$, $U_3 = {}^\#\mathtt{78}$.

To multiply α by β, do the following for $k = 0, 1, \ldots, h(\beta)-1$ (in that order), and for each letter $b \in L_k(\beta)$ (in any order): Set $j \leftarrow h(\alpha)$; then while $j > 0$ and $T(b) \mathbin{\&} U_{j-1}(\alpha) = 0$, set $j \leftarrow j - 1$. If $j = h(\alpha)$, set $L_j(\alpha)$ empty, $U_j(\alpha) \leftarrow 0$, and $h(\alpha) \leftarrow h(\alpha) + 1$. Insert b into $L_j(\alpha)$, and set $U_j(\alpha) \leftarrow U_j(\alpha) + T(b)$.

adjacent pairs of letters, avoiding
dependence graph
partial ordering
topological sortings
empilement
linear extensions, see Topological sorting
Gessel
canonical string

Viennot
starfish
line graph
claw
matchings
Möbius polynomial
matching polynomial
real roots of polynomials
cocomparability graph
Anisimov
Knuth

328. Do the following for $k = h(\beta) - 1, \ldots, 1, 0$ (in that order), and for each letter $b \in L_k(\beta)$ (in any order): Set $j \leftarrow h(\alpha) - 1$; while $j > 0$ and $T(b) \mathbin{\&} U_j(\alpha) = 0$, set $j \leftarrow j - 1$. Report failure if b isn't in $L_j(\alpha)$. Otherwise remove b from that list and set $U_j(\alpha) \leftarrow U_j(\alpha) - T(b)$; if $U_j(\alpha)$ is now zero, set $h(\alpha) \leftarrow h(\alpha) - 1$.

If there was no failure, the resulting α is the answer.

329. Do the following for $k = 0, 1, \ldots, h(\alpha) - 1$ (in that order), and for each letter $a \in L_k(\alpha)$ (in any order): Report failure if a isn't in $L_0(\beta)$. Otherwise remove a from that list, set $U_0(\beta) \leftarrow U_0(\beta) - T(a)$, and renormalize the representation of β.

Renormalization involves the following steps: Set $j \leftarrow c \leftarrow 1$. While $U_{j-1}(\beta) \neq 0$ and $c \neq 0$, terminate if $j = h(\beta)$; otherwise set $c \leftarrow 0$, $j \leftarrow j+1$, and then, for each letter b in $L_{j-1}(\beta)$ such that $T(b) \mathbin{\&} U_{j-2}(\beta) = 0$, move b from $L_{j-1}(\beta)$ to $L_{j-2}(\beta)$ and set $U_{j-2}(\beta) \leftarrow U_{j-2}(\beta) + T(b)$, $U_{j-1}(\beta) \leftarrow U_{j-1}(\beta) - T(b)$, $c \leftarrow 1$. Finally, if $U_{j-1}(\beta) = 0$, set $U_{i-1}(\beta) \leftarrow U_i(\beta)$ and $L_{i-1}(\beta) \leftarrow L_i(\beta)$ for $j \le i < h(\beta)$, then set $h(\beta) \leftarrow h(\beta) - 1$.

If there was no failure, the resulting β is the answer.

330. Let the territorial universe be $V \cup E$, the vertices plus edges of G, and let $T(a) = \{a\} \cup \{\{a,b\} \mid a \,\text{---}\, b\}$. [G. X. Viennot, in 1985, called this subgraph a *starfish*.] Alternatively, we can get by with just two elements in each set $T(a)$ if and only if $G = L(H)$ is the line graph of some other multigraph H. Then each vertex a of G corresponds to an edge $u \,\text{---}\, v$ in H, and we can let $T(a) = \{u, v\}$.

[*Notes:* The smallest graph G that *isn't* a line graph is the "claw" $K_{1,3}$. Since sets of independent vertices in the line graph G are sets of disjoint edges in H, also called *matchings* of H, the Möbius polynomial of G is also known as the "matching polynomial" of H. Such polynomials are important in theoretical chemistry and physics. When all territories have $|T(a)| \le 2$, all roots of the polynomial $M_G^*(z)$ in (149) are real and positive, by exercise 341. But $M_{\rm claw}(z,z,z,z) = 1 - 4z + 3z^2 - z^3$ has complex roots ≈ 0.317672 and $1.34116 \pm 1.16154i$.]

331. If α is a string with $k > 0$ occurrences of the substring ac, there are 2^k ways to decompose α into factors $\{a, b, c, ac\}$, and the expansion includes $+\alpha$ and $-\alpha$ each exactly 2^{k-1} times. Thus we're left with the sum of all strings that don't contain 'ac'.

332. No: If b commutes with a and c, but $ac \neq ca$, we're dealing with strings that contain no adjacent pairs ba or cb; hence cab qualifies, but it's equivalent to the smaller string bca. [Certain graphs do define traces with the stated property, as we've seen in (135) and (136). Using the next exercise we can conclude that the property holds if and only if no three letters $a < b < c$ have $a \not\,\text{---}\, b$, $b \not\,\text{---}\, c$, and $a \,\text{---}\, c$ in the graph G of clashes. Thus the letters can be arranged into a suitable linear order if and only if G is a cocomparability graph; see Section 7.4.2.]

333. To show that $\sum_{\alpha \in A, \beta \in B} (-1)^{|\beta|} \alpha\beta = 1$, let $\gamma = a_1 \ldots a_n$ be any nonempty string. If γ cannot be factored so that $a_1 \ldots a_k \in A$ and $a_{k+1} \ldots a_n \in B$, then γ doesn't appear. Otherwise γ has exactly two such factorizations, one in which k has its smallest possible value and the other in which k is one greater; these factorizations cancel each other in the sum. [*Manuscripta Math.* **19** (1976), 239–241.]

334. Equivalently we want to generate all strings of length n on the alphabet $\{1, \ldots, m\}$ that satisfy the following criterion, which strengthens the adjacent-letter test of exercise 332: If $1 \le i < j \le n$, $x_i \not\,\text{---}\, x_j$, $x_{i+1} \not\,\text{---}\, x_j$, $\ldots$, $x_{j-1} \not\,\text{---}\, x_j$, then $x_i \le x_j$. [See A. V. Anisimov and D. E. Knuth, *Int. J. Comput. Inf. Sci.* **8** (1979), 255–260.]

T1. [Initialize.] Set $x_0 \leftarrow 0$ and $x_k \leftarrow 1$ for $1 \le k \le n$.

T2. [Visit.] Visit the trace $x_1 \ldots x_n$.

T3. [Find k.] Set $k \leftarrow n$. While $x_k = m$ set $k \leftarrow k-1$. Terminate if $k=0$.

T4. [Advance x_k.] Set $x_k \leftarrow x_k + 1$ and $j \leftarrow k-1$.

T5. [Is x_k valid?] If $x_j > x_k$ and $x_j \not\!\!-\!\!- x_k$, return to T4. If $j > 0$ and $x_j < x_k$ and $x_j \not\!\!-\!\!- x_k$, set $j \leftarrow j-1$ and repeat this step.

T6. [Reset $x_{k+1} \ldots x_n$.] While $k < n$ do the following: Set $k \leftarrow k+1$, $x_k \leftarrow 1$; while $x_{k-1} > x_k$ and $x_{k-1} \not\!\!-\!\!- x_k$, set $x_k \leftarrow x_k + 1$. Then go back to T2. ▌

335. Given such an ordering, we have $M_G = \det(I - A)$, where the entry in row u and column v of A is $v[u \ge v \text{ or } u \!-\!\!-\! v]$. The determinant in the given example is

$$\det\begin{pmatrix} 1 & -b & -c & 0 & 0 & 0\\ 0 & 1-b & 0 & -d & 0 & 0\\ 0 & -b & 1-c & -d & -e & 0\\ 0 & -b & -c & 1-d & 0 & -f\\ 0 & -b & -c & -d & 1-e & -f\\ 0 & -b & -c & -d & -e & 1-f \end{pmatrix} + \det\begin{pmatrix} -a & -b & -c & 0 & 0 & 0\\ 0 & 1 & c & -d & 0 & 0\\ 0 & 0 & 1 & -d & -e & 0\\ 0 & 0 & 0 & 1-d & 0 & -f\\ 0 & 0 & 0 & -d & 1-e & -f\\ 0 & 0 & 0 & -d & -e & 1-f \end{pmatrix},$$

after expanding the first column, then subtracting the first row from all other rows in the right-hand determinant. Therefore this rule satisfies recurrence (142).

[The result also follows from MacMahon's Master Theorem, exercise 5.1.2–20, using the characterization of lexicographically smallest traces in answer 334. According to Theorem 5.1.2B, such traces are in one-to-one correspondence with multiset permutations whose two-line representation does not contain $\genfrac{}{}{0pt}{}{v}{u}$ when $v > u$ and $v \not\!\!-\!\!- u$. Is there a similar determinantal expression when G is *not* a cocomparability graph?]

336. (a) If α is a trace for G and β is a trace for H, we have $\mu_{G\oplus H}(\alpha\beta) = \mu_G(\alpha)\mu_H(\beta)$. Hence $M_{G\oplus H} = M_G M_H$. (b) In this case $\mu_{G-\!\!-H}(\alpha\beta) = \mu_G(\alpha)$ if $\beta = \epsilon$, $\mu_H(\beta)$ if $\alpha = \epsilon$; otherwise it's zero. Therefore $M_{G-\!\!-H} = M_G + M_H - 1$.

[These rules determine M_G recursively whenever G is a cograph (see exercise 7–90). In particular, complete bipartite and k-partite graphs have simple Möbius series, exemplified by $M_G = (1-a)(1-b)(1-c) + (1-d)(1-e) + (1-f) - 2$ when $G = K_{3,2,1}$.]

337. Substituting $a_1 + \cdots + a_k$ for a in M_G gives $M_{G'}$. (Each trace for G' is obtained by putting subscripts on the a's of the traces for G.)

338. The proof of Theorem F needs only minor changes: We limit α to traces that contain no elements of A, and we define α' and β' by letting a be the smallest letter $\notin A$ in the bottom level of γ's empilement. If γ has no such letter, there's only one factorization, with $\alpha = \epsilon$. Otherwise we pair up cancelling factorizations. [Incidentally, the sum of all traces whose *sinks* are in A must be written in the other order: $M_G^{-1} M_{G\setminus A}$.]

339. (a) "Push down" on piece x_j and factor out what comes through the floor.

(b) Factor out the pyramid for the smallest label, and repeat on what's left.

(c) This is a general convolution principle for labeled objects [see E. A. Bender and J. R. Goldman, *Indiana Univ. Math. J.* **20** (1971), 753–765]. For example, when $l = 3$ the number of ways to get a labeled trace of length n from three labeled pyramids is $\sum_{i,j,k} \binom{n}{i,j,k} P_i P_j P_k / 3! = n! \sum_{i,j,k} (P_i/i!)(P_j/j!)(P_k/k!)/3!$, with $i+j+k = n$ in both of these sums. We divide by 3! so that the topmost pyramid labels will be increasing.

(d) Sum the identity in (c) for $l = 0, 1, 2, \ldots$.

(e) $T(z) = \sum_{n\ge 0} t_n z^n = 1/M_G(z)$ by Theorem F, and $P(z) = \sum_{n\ge 1} p_n z^n/n$. *Note:* If we retain the letter names, writing for example $M_G(z) = 1 - (a+b+c)z + acz^2$ instead of $M_G(z) = 1 - 3z + z^2$, the formal power series $-\ln M_G(z)$ exhibits the pyramids of length n in the coefficient of z^n, but only in the sense of *commutative* algebra (not

MacMahon's Master Theorem
lexicographically smallest traces
multiset permutations
cocomparability graph
cograph
complete bipartite
complete k-partite graphs
convolution principle
binomial convolution
Bender
Goldman

trace algebra). For example, the coefficient of z^3 obtained from $\sum_{k\ge1}(1-M_G(z))^k/k$ with trace algebra includes the nonpyramidal term $bac/6$.

permanent
Heilmann
Lieb
determinant
MacMahon's Master Theorem
Scott
Sokal
slack

340. Let $w((i_1 \ldots i_k)) = (-1)^{k-1}a_{i_1i_2}a_{i_2i_3}\ldots a_{i_ki_1}$; thus $w(\pi) = (-a_{13}a_{34}a_{42}a_{21})(-a_{57}a_{75})(a_{66})$ in the given example. The permutation polynomial is then $\det A$, by definition of the determinant. (And we get the *permanent*, if we omit the $(-1)^{k-1}$.)

341. The hint is true when $n=2$, since the first involution polynomials are $w_{11}x$ and $w_{11}w_{22}x^2 - w_{12}$. And there's a recurrence: $W(S) = w_{ii}xW(S\setminus i) - \sum_{j\ne i}W(S\setminus\{i,j\})$.

So we can prove the existence of $n+1$ roots $s_1 < r_1 < \cdots < r_n < s_{n+1}$ by induction: Let $W_n(x)$ be the polynomial for $\{1,\ldots,n\}$. Then $W_{n+1}(x)$ is $w_{(n+1)(n+1)}xW_n(x)$ minus n polynomials $w_{(n+1)j}W(\{1,\ldots,n\}\setminus j)$, each with roots $q_k^{(j)}$ that are nicely sandwiched between the roots of W_n. Furthermore $q_{n-k}^{(j)} = -q_k^{(j)}$ and $r_{n+1-k} = -r_k$, for $1 \le k \le n/2$. It follows that $W_{n+1}(r_n) < 0$, $W_{n+1}(r_{n-1}) > 0$, and so on, with $(-1)^kW_{n+1}(r_{n+1-k}) > 0$ for $1 \le k \le n/2$. Moreover, $W_{n+1}(0) = 0$ when n is even; $(-1)^kW_{n+1}(0) > 0$ when $n = 2k-1$; and $W_{n+1}(x) > 0$ for all large x. Hence the desired s_k exist. [See Heilmann and Lieb, *Physical Review Letters* **24** (1970), 1412.]

342. If we replace $(i_1 \ldots i_k)$ by $a_{i_1i_2}a_{i_2i_3}\ldots a_{i_ki_1}$ (as in answer 340, but without the $(-1)^{k-1}$), then M_{G_n} becomes $\det(I-A)$. Replacing a_{ij} by $a_{ij}x_j$ gives the determinant in MacMahon's Master Theorem. And if $x_1 = \cdots = x_n = x$, we get the polynomial $\det(I-xA)$, whose roots are the reciprocals of the roots of A's characteristic polynomial.

343. The formulas in answer 336 show that $M_G(p_1,\ldots,p_m)$ increases whenever any p_j decreases, with respect to a cograph G. The only graph on ≤ 4 vertices that isn't a cograph is P_4 (see exercise 7–90); then $M_G(p_1,p_2,p_3,p_4) = 1-p_1-p_2-p_3-p_4+p_1p_3+p_1p_4+p_2p_4 = (1-p_1)(1-p_3-p_4)-p_2(1-p_4)$. In this case also we can conclude that $M_G(p_1,\ldots,p_4) > 0$ implies $(p_1,\ldots,p_4)\in\mathcal{R}(G)$. But when $G = P_5$, we find $M_G(1-\epsilon,1-\epsilon,\epsilon,1-\epsilon,1-\epsilon) > 0$ for $0 \le \epsilon < \phi^{-2}$; yet $(1-\epsilon,1-\epsilon,\epsilon,1-\epsilon,1-\epsilon)$ is never in $\mathcal{R}(G)$ because $M_G(0,0,\epsilon,1-\epsilon,1-\epsilon) = -(1-\epsilon)^2$.

344. (a) If some minterm, say $B_1\overline{B}_2\overline{B}_3B_4$, has negative "probability," then $p_1p_4 \times (1-\pi_2-\pi_3+\pi_{23}) < 0$; hence $M_G(0,p_2,p_3,0) < 0$ violates the definition of $\mathcal{R}(G)$.

(b) In fact, more is true: $\pi_{I\cup J} = \pi_I\pi_J$ when $i \not\!\!-\!\!\!- j$ for $i\in I$, $j\in J$, and $I\cap J=\emptyset$.

(c) It's $M_G(p_1[1\in J],\ldots,p_m[m\in J])$, by (140) and (141). This important fact, already implicit in the solution to part (a), implies that $\beta(G\mid J) > 0$ for all J.

(d) Writing just 'J' for '$G|J$', we shall prove that $\alpha(i\cup J)/\beta(i\cup J) \ge \alpha(J)/\beta(J)$ for $i\notin J$, by induction on $|J|$. Let $J' = \{j\in J \mid i \not\!\!-\!\!\!- j\}$. Then we have

$$\alpha(i\cup J) = \alpha(J) - \Pr\Big(A_i \cap \bigcap_{j\in J}\overline{A}_j\Big) \ge \alpha(J) - \Pr\Big(A_i \cap \bigcap_{j\in J'}\overline{A}_j\Big) \ge \alpha(J) - p_i\alpha(J'),$$

because of (133). Also $\beta(i\cup J) = \beta(J) - p_i\beta(J')$. Hence $\alpha(i\cup J)\beta(J) - \alpha(J)\beta(i\cup J) \ge (\alpha(J)-p_i\alpha(J'))\beta(J) - \alpha(J)(\beta(J)-p_i\beta(J')) = p_i(\alpha(J)\beta(J') - \alpha(J')\beta(J))$, which is ≥ 0 by induction since $J'\subseteq J$.

[This argument proves that Lemma L holds whenever $(p_1,\ldots,p_m)$ leads to a legitimate probability distribution with $\beta(G) > 0$; hence such probabilities are in $\mathcal{R}(G)$.]

(e) By induction, we have $\beta(i\cup J) = \beta(J) - \theta_i\beta(J')\prod_{i \,-\!\!\!-\, j}(1-\theta_j) \ge \beta(J) - \theta_i\beta(J')\prod_{j\in J\setminus J'}(1-\theta_j) \ge (1-\theta_i)\beta(J)$, because $\beta(J)/\beta(J') \ge \prod_{j\in J\setminus J'}(1-\theta_j)$.

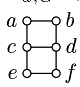
345. (Solution by A. D. Scott and A. D. Sokal.) Set $p'_j = (1+\delta)p_j$ where $\delta \le 0$ is the slack of $(p_1,\ldots,p_m)$. Then $M_G(p'_1,\ldots,p'_m) = 0$, but it becomes positive if any p'_j is decreased. Define events B'_1, …, B'_m by the construction in exercise 344. Let C_1, …, C_m be independent binary random variables such that $\Pr(C_j = 1) = q_j$,

traces
sources
top-down algs
bottom-up algs
Scott
Sokal

where $(1-p_j')(1-q_j) = 1-p_j$. Then the events $B_j = B_j' \vee C_j$ satisfy the desired conditions: $\Pr(B_i \mid \overline{B}_{j_1} \cap \cdots \cap \overline{B}_{j_k}) = \Pr(B_i \mid \overline{B}'_{j_1} \cap \cdots \cap \overline{B}'_{j_k}) = \Pr(B_i) = p_i$; and $\Pr(B_1 \vee \cdots \vee B_m) \geq \Pr(B_1' \vee \cdots \vee B_m') = 1$.

346. (a) By (144), $K_{a,G}$ is the sum of all traces on the probabilities of $G \setminus a$ whose sources are neighbors of a. Decreasing p_j doesn't decrease any trace.

(b) Suppose vertex $a = 1$ has neighbors $2, \ldots, j$. If we've recursively computed $M_{G\setminus a^*}$ and $M_{G\setminus a}$, finding that $(p_{j+1}, \ldots, p_m) \in \mathcal{R}(G\setminus a^*)$ and $(p_2, \ldots, p_m) \in \mathcal{R}(G\setminus a)$, then we know $K_{a,G}$; and the monotonicity property in (a) implies that $(p_1, \ldots, p_m) \in \mathcal{R}(G)$ if and only if $a K_{a,G} < 1$.

The graph $G =$  in exercise 335 can, for example, be processed as follows:

$$\begin{aligned}
M_{abcdef} &= M_{bcdef}\Bigl(1-a\frac{M_{def}}{M_{bcdef}}\Bigr) = (1-a')(1-b')\ldots(1-f'), & a' &= \frac{a}{(1-b')(1-c')},\\
M_{bcdef} &= M_{cdef}\Bigl(1-b\frac{M_{cef}}{M_{cdef}}\Bigr) = (1-b')(1-c')\ldots(1-f'), & b' &= \frac{b(1-c'')}{(1-c')(1-d')},\\
M_{cdef} &= M_{def}\Bigl(1-c\frac{M_{f}}{M_{def}}\Bigr) = (1-c')(1-d')(1-e')(1-f'), & c' &= \frac{c}{(1-d')(1-e')},\\
M_{cef} &= M_{ef}\Bigl(1-c\frac{M_{f}}{M_{ef}}\Bigr) = (1-c'')(1-e')(1-f'), & c'' &= \frac{c}{(1-e')},\\
M_{def} &= M_{ef}\Bigl(1-d\frac{M_{e}}{M_{ef}}\Bigr) = (1-d')(1-e')(1-f'), & d' &= \frac{d(1-e'')}{(1-e')(1-f')},\\
M_{ef} &= M_{f}\Bigl(1-e\frac{M_{\epsilon}}{M_{f}}\Bigr) = (1-e')(1-f'), & e' &= \frac{e}{(1-f')},\\
M_{e} &= M_{\epsilon}\Bigl(1-e\frac{M_{\epsilon}}{M_{\epsilon}}\Bigr) = (1-e''), & e'' &= e,\\
M_{f} &= M_{\epsilon}\Bigl(1-f\frac{M_{\epsilon}}{M_{\epsilon}}\Bigr) = (1-f'), & f' &= f,
\end{aligned}$$

with $M_\epsilon = 1$. (The equations on the left are derived top-down, then the equations on the right are evaluated bottom-up. We have $(a, b, \ldots, f) \in \mathcal{R}(G)$ if and only if $f' < 1$, $e'' < 1$, $e' < 1$, …, $a' < 1$.) Even better is to traverse this graph in another order, using the rule $M_{G\oplus H} = M_G M_H$ (exercise 336) when subgraphs aren't connected:

$$\begin{aligned}
M_{cdabef} &= M_{dabef}\Bigl(1-c\frac{M_b M_f}{M_{dabef}}\Bigr) = (1-c')(1-d')\ldots(1-f'), & c' &= \frac{c}{(1-a')(1-d')(1-e')},\\
M_{dabef} &= M_{ab}M_{ef}\Bigl(1-d\frac{M_a M_e}{M_{ab}M_{ef}}\Bigr) = (1-d')(1-a')(1-b')(1-e')(1-f'), & & \text{(see below)}\\
M_{ab} &= M_{b}\Bigl(1-a\frac{M_{\epsilon}}{M_{b}}\Bigr) = (1-a')(1-b'), & a' &= \frac{a}{(1-b')},\\
M_{a} &= M_{\epsilon}\Bigl(1-a\frac{M_{\epsilon}}{M_{\epsilon}}\Bigr) = (1-a''), & a'' &= a,\\
M_{b} &= M_{\epsilon}\Bigl(1-b\frac{M_{\epsilon}}{M_{\epsilon}}\Bigr) = (1-b'), & b' &= b,
\end{aligned}$$

where $d' = dM_a M_\epsilon/(M_{ab}M_{ef}) = d(1-a'')(1-e'')/((1-a')(1-b')(1-e')(1-f'))$, and M_{ef}, M_e, M_f, M_ϵ are as before. In this way we can often solve the problem in linear time. [See A. D. Scott and A. D. Sokal, *J. Stat. Phys.* **118** (2005), 1151–1261, §3.4.]

least common ancestor
Pegden

347. (a) Suppose $v_1 — v_2 — \cdots — v_k — v_1$ is an induced cycle. We can assume that $v_1 \succ v_2$. Then, by induction on j, we must have $v_1 \succ \cdots \succ v_j$ for $1 < j \le k$; for if $v_{j+1} \succ v_j$ we would have $v_{j+1} — v_{j-1}$ by (∗). But now $v_k — v_1$ implies that $k = 3$.

(b) Let the vertices be $\{1, \ldots, m\}$, with territory sets $T(a) \subseteq U$ for $1 \le a \le m$; and let U be a tree such that each $U \mid T(a)$ is connected. Let U_a be the least common ancestor of $T(a)$ in U. (Thus the nodes of $T(a)$ appear at the top of the subtree rooted at U_a.) Since $U_a \in T(a)$, we have $a — b$ when $U_a = U_b$.

Writing $s \succ_U t$ for the ancestor relation in U, we now define $a \succ b$ if $U_a \succ_U U_b$ or if $U_a = U_b$ and $a < b$. Then (∗) is satisfied: If $t \in T(a) \cap T(b)$, we have $U_a \succ_U t$ and $U_b \succ_U t$, hence $U_a \succeq_U U_b$ or $U_b \succeq_U U_a$, hence $a \succ b$ or $b \succ a$. And if $a \succ b \succ c$ and $t \in T(a) \cap T(c)$, we have $U_a \succeq_U U_b \succeq_U U_c$; consequently $U_b \in T(a) \cap T(b)$, because U_b lies on the unique path between t and U_a in U and $T(a)$ is connected.

(c) Processing the nodes in any order such that a is eliminated before b whenever U_a is a proper ancestor of U_b will then lead only to subproblems in which the algorithm needs no "double-primed" variables.

For example, using $(a, b, \ldots, g)$ instead of $(1, 2, \ldots, 7)$ in order to match the notation in exercise 346, suppose U is the tree rooted at p having the edges $p — q$, $p — r$, $r — s$, $r — t$, and let $T(a) = \{p, q, r, t\}$, $T(b) = \{p, r, s\}$, $T(c) = \{p, q\}$, $T(d) = \{q\}$, $T(e) = \{r, s\}$, $T(f) = \{s\}$, $T(g) = \{t\}$. Then $a \succ b \succ c \succ d$, $c \succ e \succ f$, $e \succ g$. The algorithm computes $M_{abcdefg} = (1 - a')M_{bcdefg}$, $M_{bcdefg} = (1 - b')M_{cdefg}$, etc., where $a' = aM_f / M_{bcdefg}$, $b' = bM_{dfg}/M_{cdefg} = b(M_d M_f M_g)/(M_{cd} M_{ef} M_g)$, etc.

In general, the tree ordering guarantees that no "double-primed" variables are needed. Thus the formulas reduce to $v' = v / \prod_{u — v,\, v \succ u} (1 - u')$ for each vertex v.

(d) For example, we have $p_1 = a, \ldots, p_7 = g$, $\theta_1 = a', \ldots, \theta_7 = g'$ in (c). The values of the θ's, which depend on the ordering $\succ$, are uniquely defined by the given equations; and we have $M_G(p_1, \ldots, p_m) = (1 - \theta_1) \ldots (1 - \theta_m)$ in any case. [W. Pegden, *Random Structures & Algorithms* **41** (2012), 546–556.]

348. There is at least one singularity at $z = \rho e^{i\theta}$ for some θ. If $0 < r < \rho$, the power series $f(z) = \sum_{n=0}^{\infty} f^{(n)}(re^{i\theta})(z - re^{i\theta})^n / n!$ has radius of convergence $\rho - r$. If $z = \rho$ isn't a singularity, the radius of convergence for $\theta = 0$ would exceed $\rho - r$. But $|f^{(n)}(re^{i\theta})| = \left|\sum_{m=0}^{\infty} m^{\underline{n}} a_n (re^{i\theta})^{m-n}\right| \le f^{(n)}(r)$. [*Mathematische Annalen* **44** (1894), 41–42.]

349. Typical generating functions are $g_{0000001} = 1$; $g_{0110110} = z(g_{0100110} + g_{0101110} + g_{0110110} + g_{0111110})/4$ (in Case 1) or $g_{0110110} = z(g_{0000110} + g_{0010110} + g_{0100110} + g_{0110110})/4$ (in Case 2). These systems of 128 linear equations have solutions whose denominators involve one or more of the polynomials $4 - z$, $2 - z$, $16 - 12z + z^2$, $4 - 3z$, $64 - 80z + 24z^2 - z^3$, $8 - 8z + z^2$ in Case 1 (see exercise 320); the denominators in Case 2 are powers of $4 - z$.

Setting $g(z) = \sum_x g_x(z)/128$ leads to $g(z) = 1/((2 - z)(8 - 8z + z^2))$ in Case 1, with mean 7 and variance 42; $g(z) = (1088 - 400z + 42z^2 - z^3)/(4 - z)^6$ in Case 2, with mean $1139/729 \approx 1.56$ and variance $1139726/729^2 \approx 2.14$.

[The upper bound $E_1 + \cdots + E_6$ is achieved by the distribution of Case 1, because it matches the extreme distribution (148) of the path graph P_6. Incidentally, if Case 1 is generalized from $n = 7$ to arbitrary n, the mean is $n(n-1)/6$ and the variance is $(n + 3)(n + 2)n(n - 1)/90$.]

350. (a) The generating function for N is $\prod_{k=1}^{n} (1 - \xi_k)/(1 - \xi_k z)$; so the mean and variance, in general, are $\sum_{k=1}^{n} \xi_k/(1 - \xi_k)$ and $\sum_{k=1}^{n} \xi_k/(1 - \xi_k)^2$. In particular, the means are (i) n; (ii) $n/(2n - 1)$; (iii) $n/(2^n - 1)$; (iv) $H_{2n} - H_n + \frac{1}{2n} = \ln 2 + O(1/n)$; (v) $\frac{1}{2}\left(\frac{1}{n+1} + \frac{1}{2n} - \frac{1}{2n+1}\right) = \frac{1}{2n} + O(1/n^2)$. The variance in case (i) is 2; otherwise it's asymptotically the same as the mean, times $1 + O(1/n)$.

(b) In this case the mean and variance are $\xi/(1-\xi)$ and $\xi/(1-\xi)^2$, where $\xi = \Pr(A_m) = 1-(1-\xi_1)\ldots(1-\xi_n)$. This value ξ is (i) $1-2^{-n}$; (ii) $1-\bigl(1-\frac{1}{2n}\bigr)^n = 1-e^{-1/2}+O(1/n)$; (iii) $1-(1-2^{-n})^n = n/2^n + O(n^2/4^n)$; (iv) $1/2$; (v) $1/(2n+2)$. Hence the respective means are (i) 2^n-1; (ii) $e^{1/2}-1+O(1/n)$; (iii) $n/2^n+O(n^2/4^n)$; (iv) 1; (v) $1/(2n+1)$. And the variances are (i) 4^n-2^n; (ii) $e-e^{1/2}+O(1/n)$; (iii) $n/2^n+O(n^2/4^n)$; (iv) 2; (v) $1/(2n+1)+1/(2n+1)^2$.

(c) Since G is $K_{n,1}$, exercises 336 and 343 imply that $(\xi_1,\ldots,\xi_n,\xi)\in\mathcal{R}(G)$ if and only if $\xi<\frac12$. This condition holds in cases (ii), (iii), and (v).

351. (Solution by Moser and Tardos.) We require $i \mathrel{—} j$ if there's a setting of the variables such that A_i is false and A_j is true, provided that some change to the variables of Ξ_j might make A_i true. And vice versa with $i\leftrightarrow j$.

(The Local Lemma can be proved also for *directed* lopsidependency graphs; see Noga Alon and Joel H. Spencer, *The Probabilistic Method* (2008), §5.1. But the theory of traces, which we use to analyze Algorithm M, is based on undirected graphs, and no algorithmic extension to the directed case is presently known.)

352. Answer 344(e), with $M_G=\beta(i\cup J)$, $M_{G\setminus i}=\beta(J)$, proves that $M_{G\setminus i}/M_G\ge 1-\theta_i$.

353. (a) There are $n+1$ sorted strings in Case 1, namely 0^k1^{n-k} for $0\le k\le n$. There are F_{n+2} solutions in Case 2 (see, for example, exercise 7.2.1.1–91).

(b) At least $2^nM_G(1/4)$, where G is the path P_{n-1}. By exercise 320 we have $M_G(1/4)=f_{n-1}(1/4)=(n+1)/2^n$; so Case 1 matches the lower bound.

(c) There are *no* lopsidependencies. Hence the relevant G is the empty graph on $m=n-1$ vertices; $M_G(1/4)=(3/4)^{n-1}$ by exercise 336; and indeed, $F_{n+2}\ge 3^{n-1}2^{2-n}$.

354. Differentiate (151) and set $z\leftarrow 1$.

355. If $A=A_j$ is an isolated vertex of G, then $1-p_jz$ is a factor of the polynomial $M_G^*(z)$ in (149), hence $1+\delta\le 1/p_j$; and $E_j=p_j/(1-p_j)\le 1/\delta$. Otherwise $M_G(p_1,\ldots,p_{j-1},p_j(1+\delta),p_{j+1},\ldots,p_m)=M_G^*(1)-\delta p_jM_{G\setminus A^*}^*(1)>M_G^*(1+\delta)=0$; so $E_j=p_jM_{G\setminus A^*}^*(1)/M_G^*(1)>1/\delta$.

356. (a) We prove the hint by induction on $|S|$. It's obvious when $S=\emptyset$; otherwise let $X=S\cap\bigcup_{i\in U_j}U_j$ and $Y=S\setminus X$. We have

$$\Pr(A_i\mid\overline{A}_S)=\frac{\Pr(A_i\cap\overline{A}_X\cap\overline{A}_Y)}{\Pr(\overline{A}_X\cap\overline{A}_Y)}\le\frac{\Pr(A_i\cap\overline{A}_Y)}{\Pr(\overline{A}_X\cap\overline{A}_Y)}\le\frac{\Pr(A_i)\Pr(\overline{A}_Y)}{\Pr(\overline{A}_X\cap\overline{A}_Y)}=\frac{\Pr(A_i)}{\Pr(\overline{A}_X\mid\overline{A}_Y)}$$

by (133). Suppose i belongs to the cliques $U_{j_0},\ldots,U_{j_r}$ where $j=j_0$. Let $X_0=\emptyset$ and $X_k=(S\cap U_{j_k})\setminus X_{k-1}$, $Y_k=Y\cup X_1\cup\cdots\cup X_{k-1}$ for $1\le k\le r$. We have $\Pr(A_l\mid\overline{A}_{Y_k})\le\theta_{lj_k}$ for all $l\in X_k$, since $|Y_k|<|S|$ when $X_k\ne\emptyset$; hence $\Pr(\overline{A}_{X_k}\mid\overline{A}_{Y_k})\ge(1+\theta_{ij_k}-\Sigma_{j_k})$. Thus $\Pr(\overline{A}_X\mid\overline{A}_Y)=\Pr(\overline{A}_{X_1}\mid\overline{A}_{Y_1})\Pr(\overline{A}_{X_2}\mid\overline{A}_{Y_2})\ldots\Pr(\overline{A}_{X_r}\mid\overline{A}_{Y_r})\ge\prod_{k\ne j,i\in U_k}(1+\theta_{ik}-\Sigma_k)$, by the chain rule (exercise MPR-14); the hint follows.

Finally let $W_k=U_1\cup\cdots\cup U_k$ for $1\le k\le t$. The hint implies that

$$\begin{aligned}\Pr(\overline{A}_1\cap\cdots\cap\overline{A}_m)&=\Pr(\overline{A}_{W_1})\Pr(\overline{A}_{W_2}\mid\overline{A}_{W_1})\ldots\Pr(\overline{A}_{W_t}\mid\overline{A}_{W_{t-1}})\\&\ge(1-\Sigma_1)(1-\Sigma_2)\ldots(1-\Sigma_t)>0.\end{aligned}$$

(b) The extreme events $B_1,\ldots,B_m$ of Theorem S satisfy the hint of (a). Thus $\Pr(B_i\mid\bigcap_{k\notin U_j}\overline{B}_k)\le\theta_{ij}$ for all $i\in U_j$; hence $q_i=\Pr(B_i\mid\bigcap_{k\ne i}\overline{B}_k)\le\theta_{ij}/(1+\theta_{ij}-\Sigma_j)$. Furthermore $E_i=q_i/(1-q_i)$ in (152), because $q_i=p_iM_{G\setminus i^*}/M_{G\setminus i}$.

complete bipartite graph
Moser
Tardos
Alon
Spencer
traces
Fibonacci numbers
consecutive ones
chain rule

Kolipaka
Szegedy
Xu
covering
invariant property
cufflink pattern
L_7
lower semimodular
modular

(c) Let $U_1, \ldots, U_t$ be the *edges* of G, with $\theta_{ik} = \theta_i$ when $U_k = \{i, j\}$. Then $\Sigma_k = \theta_i + \theta_j < 1$, and the sufficient condition in (a) is that $\Pr(A_i) \le \theta_i \prod_{j \ne k, i \text{---} j}(1-\theta_j)$ whenever $i \text{---} k$. (But notice that Theorem M does not hold for such larger p_i.)

[K. Kolipaka, M. Szegedy, and Y. Xu, *LNCS* **7408** (2012), 603–614.]

357. If $r > 0$, we have $x = r/(1-p)$, $y = r/(1-q)$. But $r = 0$ is possible only on the axes of Fig. 51: Either $(p, q) = (0, 1)$, $x = 0$, $0 < y \le 1$, or $(p, q) = (1, 0)$, $0 < x \le 1$, $y = 1$.

358. Suppose $x \ge y$ (hence $p \ge q$ and $x > 0$). Then $p \le r$ if and only if $1 - y \le y$.

359. Instead of computing π_l by formula (154), represent it as two numbers (π_l^+, π_l'), where π_l^+ is the product of the nonzero factors and π_l' is the number of zero factors. Then the quantity $\pi_{\bar l}$ needed in (156) is $\pi_{\bar l}^+[\pi_{\bar l}' = 0]$; and the quantity $\pi_l/(1 - \eta_{C \to l})$ is $\pi_l^+[\pi_l' = 1]$ if $\eta_{C \to l} = 1$, otherwise it's $\pi_l^+[\pi_l' = 0]/(1 - \eta_{C \to l})$. A similar method can be used to separate out the zero factors of $\prod_{l \in C} \gamma_{l \to C}$ in (157).

360. We may assume that $\eta_3 = 0$. Since $\pi_l = 1$ implies that $\eta_{C \to l} = \gamma_{\bar l \to C} = 0$, we have $\eta_{C \to 1} = \eta_{C \to \bar 2} = \eta_{C \to 3} = \eta_{C \to \bar 4} = \gamma_{\bar 1 \to C} = \gamma_{2 \to C} = \gamma_{\bar 3 \to C} = \gamma_{4 \to C} = 0$ for all C. Consequently, as in (159), all but three of the values $\eta_{C \to l}$ are zero; let x, y, z denote the others. Also let $\eta_{\bar 1} = a$, $\eta_2 = b$, $\eta_4 = c$, $\eta_{\bar 3} = d$. Then $\pi_{\bar 1} = (1-a)(1-x)$, $\pi_2 = (1-b)(1-y)$, $\pi_4 = (1-c)(1-z)$, and $\pi_{\bar 3} = 1 - d$. A fixed point is obtained if $x = d(b + cd(1-b) + ad^2(1-b)(1-c))/(1 - d^3(1-a)(1-b)(1-c))$, etc. If d is 0 or 1 then $x = y = z = d$. [Are there any other fixed points, say with $\pi_1 \ne 1$?]

361. The π's and γ's will also be either 0 or 1, and we exclude the case $\pi_l = \pi_{\bar l} = 0$; thus each variable v is either 1, 0, or $*$, depending on whether $(\pi_v, \pi_{\bar v})$ is $(0, 1)$, $(1, 0)$, or $(1, 1)$.

Any assignment of 1, 0, or $*$ to the variables is permissible, provided that every clause has at least one literal that's true or two that are $*$. (Such partial assignments are called "covering," and they're usually possible even with unsatisfiable clauses; see exercise 364.) All survey messages $\eta'_{C \to l} = \eta_{C \to l}$ are zero except when clause C has l as its only non-false literal. The reinforcement message η_l can be either 0 or 1, except that it must be 1 if l is true ($\pi_l = 0$) and all messages $\eta_{C \to l}$ are 0.

If we also want $\eta_l' = \eta_l$, we take $\kappa = 1$ in (158), and $\eta_l = 1 - \pi_l$.

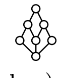
362. Create a linked list L, containing all literals that are to be forced true, including all literals that are in 1-clauses of the original problem. Do the following steps while L is nonempty: Remove a literal l from L; remove all clauses that contain l; and remove $\bar l$ from all the clauses that remain. If any of those clauses has thereby been reduced to a single literal, (l'), check to see if l' or $\bar l'$ is already present in L. If $\bar l'$ is present, a contradiction has arisen; we must either terminate unsuccessfully or restart step S8 with increased ψ. But if $\bar l'$ and l' are both absent, put l' into L.

363. (a) True; indeed, this is an important invariant property of Algorithm C.

(b) $W(001) = 1$, $W({*}{*}{*}) = p_1p_2p_3$, otherwise $W(x) = 0$.

(c) Statements (i) and (iii) are true, but not (ii); consider $x = 10{*}$, $x' = 00{*}$, and the clause 123.

(d) All eight subsets of $\{1, \bar 2, \bar 3\}$ are stable except $\{\bar 2, \bar 3\}$, because x_1 is constrained in 100. The other seven are partially ordered as shown. (This diagram illustrates L_7, the smallest lattice that is lower semimodular but not modular.)



(e)

$x_2x_3 =$	00	01	$0*$	10	11	$1*$	$*0$	$*1$	$**$
$x_1 = 0$	0	q_1q_2	0	q_1q_3	$q_1q_2q_3$	$q_1q_2p_3$	0	$q_1p_2q_3$	$q_1p_2p_3$
$x_1 = 1$	q_2q_3	$q_1q_2q_3$	$q_1q_2p_3$	$q_1q_2q_3$	$q_1q_2q_3$	$q_1q_2p_3$	$q_1p_2q_3$	$q_1p_2q_3$	$q_1p_2p_3$
$x_1 = *$	0	$p_1q_2q_3$	$p_1q_2p_3$	$p_1q_2q_3$	$p_1q_2q_3$	$p_1q_2p_3$	$p_1p_2q_3$	$p_1p_2q_3$	$p_1p_2p_3$

(f) One solution is $\{\bar1\bar23\bar4\bar5, \bar14, \bar25, \bar34\bar5, \bar3\bar45\}$. (For these clauses the partial assignment $\{3\}$ is stable, but it is "unreachable" below $\{1,2,3,4,5\}$.)

(g) If $L = L' \setminus l$ and $L' \in \mathcal{L}$ but $L \notin \mathcal{L}$, introduce the clause $(x_l \vee \bigvee_{k\in L'} \bar x_k)$.

(h) True, because $L' = L \setminus l'$ and $L'' = L \setminus l''$, where $|l'|$ and $|l''|$ are unconstrained with respect to L. A variable that's unconstrained with respect to L is also unconstrained with respect to any subset of L.

(i) Suppose $L' = L'^{(0)} \prec \cdots \prec L'^{(s)} = \{1,\ldots,n\}$ and $L'' = L''^{(0)} \prec \cdots \prec L''^{(t)} = \{1,\ldots,n\}$. Then $L'^{(s-i)} \cap L''^{(t-j)}$ is stable for $0 \le i \le s$ and $0 \le j \le t$, by induction on $i+j$ using (h).

(j) It suffices to consider the case $L = \{1,\ldots,n\}$. Suppose the unconstrained variables are x_1, x_2, x_3. Then, by induction, the sum is $q_1q_2q_3 + p_1 + p_2 + p_3 - (p_1p_2 + p_1p_3 + p_2p_3) + p_1p_2p_3 = 1$, using "inclusion and exclusion" to compensate for terms that are counted more than once. A similar argument works with any number of unconstrained variables.

Notes: See F. Ardila and E. Maneva, *Discrete Mathematics* **309** (2009), 3083–3091. The sum in (j) is ≤ 1 when each $p_k + q_k \le 1$ for $1 \le k \le n$, because it is monotone. Because of (i), the stable sets below L form a lower semimodular lattice, with

$$L' \wedge L'' = L' \cap L'' \quad\text{and}\quad L' \vee L'' = \bigcap\{L''' \mid L''' \supseteq L' \cup L'' \text{ and } L''' \sqsubseteq L\}.$$

E. Maneva and A. Sinclair noted in *Theoretical Comp. Sci.* **407** (2008), 359–369 that a random satisfiability problem is satisfiable with probability $\le \mathrm{E}\sum W(X)$, the expected total weight of partial assignments having the given distribution, because of identity (j); this led them to sharper bounds than had previously been known.

364. (a) True if and only if all clauses have length 2 or more.

(b) 001 and $***$ are covering; these are the partial assignments of nonzero weight, when $q_1 = \cdots = q_n = 0$ in the previous exercise. Only 001 is a core.

(c) $***$ is the only covering and the only core; $W(0101) = W(0111) = q_3$.

(d) In fact, every stable partial assignment L' has a unique covering assignment L with $L \sqsubseteq L'$, namely $L = \bigcap\{L'' \mid L'' \sqsubseteq L'$, obtained by successively removing unconstrained literals (in any order)$\}$.

(e) If L' and L'' are adjacent we have $L' \cap L'' \sqsubseteq L'$ and $L' \cap L'' \sqsubseteq L''$.

(f) Not necessarily. For example, the clauses $\{\bar1\bar234, \bar12\bar34, \bar123\bar4, 1\bar2\bar34, 1\bar23\bar4, 12\bar3\bar4\}$ define $\overline{S}_2(x_1,x_2,x_3,x_4)$; there are two clusters but only an empty core.

[A. Braunstein and R. Zecchina introduced the notion of covering assignments in *J. Statistical Mechanics* (June 2004), P06007:1–18.]

365. If L is any of the six solutions in (8), and if q is odd, then $qL-d$ is a covering assignment for $0 \le d < q$ and $8q-d \le n < 9q-d$. (For example, if $L = \{\bar1, \bar2, 3, 4, \bar5, \bar6, 7, 8\}$ the partial assignment $3L-1 = \{\bar2, \bar5, 8, 11, \overline{14}, \overline{17}, 20, 23\}$ works for $n \in [23\,..\,25]$.) Thus all $n > 63$ are "covered." [Do all nonempty coverings of *waerden*$(3,3;n)$ have this form?]

366. Eliminating variable 1 (x_1) by resolution yields the erp rule $\bar x_1 \leftarrow (x_2 \vee \bar x_3) \wedge (x_3 \vee x_4)$, and new clauses $\{2\bar34, 2\bar3\bar4, 234, \bar234\}$. Then eliminating 2 ($x_2$) yields $x_2 \leftarrow (x_3 \vee x_4) \wedge (\bar x_3 \vee x_4)$ and new clauses $\{34, \bar34\}$. Now 4 (x_4) is pure; so $x_4 \leftarrow 1$, and $F' = \emptyset$ is satisfiable. (Going backwards in the erp rules will then make $x_4 \leftarrow 1$, $x_2 \leftarrow 1$, $x_1 \leftarrow 0$, regardless of x_3.)

367. (We can choose whichever of the two assignments is most convenient, for example by picking the shortest, since either one is a valid erp rule.) Any solution will either satisfy all the clauses on the right side of $\bar x$ or all the clauses on the right side of x, or both. For if a solution falsifies both $C_i \setminus x$ and $C'_j \setminus \bar x$, it falsifies $C_i \diamond C'_j$.

inclusion and exclusion
Ardila
semimodular lattice
Maneva
Sinclair
symmetric Boolean functions S_k
Braunstein
Zecchina
Eliminating
resolution
pure

In either case the value of x will satisfy all of the clauses $C_1, \ldots, C_a, C'_1, \ldots, C'_b$.

368. If (l) is a clause, subsumption removes all other clauses that contain l. Then resolution (with $p = 1$) will remove $\bar{l}$ from all q of its clauses, and (l) itself.

369. Let $C_i = (l \vee \alpha_i)$ and $C'_j = (\bar{l} \vee \beta_j)$. Each omitted clause $C_i \diamond C'_j = (\alpha_i \vee \beta_j)$, where $1 < i \le p$ and $r < j \le q$, is redundant, because it is a consequence of the non-omitted clauses $(\alpha_i \vee \bar{l}_1), \ldots, (\alpha_i \vee \bar{l}_r), (l_1 \vee \cdots \vee l_r \vee \beta_j)$ via hyperresolution. [This technique is called "substitution," because we essentially replace $|l|$ by its definition.]

370. $(a \vee b) \wedge (a \vee \bar{b} \vee \bar{c}) \wedge (\bar{a} \vee b \vee c) = (a \vee \bar{c}) \wedge (b \vee c)$. (See the discussion following 7.1.1–(27). In general, advanced preprocessors use the theory of DNF minimization, in its dual form, to find irredundant minimum forms for CNF. Such techniques are not implemented, however, in the examples of preprocessing considered in this section.)

371. One scenario starts by eliminating variable 1, replacing eight clauses by eight new ones: $23\bar4\bar7$, $\bar2\bar347$, $23\bar5\bar9$, $\bar2\bar359$, $3\bar45\bar7$, $\bar34\bar57$, $4\bar57\bar9$, $\bar4\bar579$. Then 8 is eliminated, replacing another eight by eight: $2\bar45\bar6$, $\bar24\bar56$, $25\bar6\bar7$, $\bar2\bar567$, $25\bar7\bar9$, $\bar2\bar579$, $46\bar7\bar9$, $\bar4\bar679$. Then come self-subsumptions: $23\bar4\bar7 \mapsto 23\bar7$ (via 234), $3\bar45\bar7 \mapsto 35\bar7$ (345), $357 \mapsto 35$ ($35\bar7$); and 35 subsumes 345, $35\bar7$. Further self-subsumptions yield $23\bar5\bar9 \mapsto 23\bar9$, $\bar2\bar359 \mapsto \bar2\bar39$, $\bar2\bar579 \mapsto \bar279$, $\bar24\bar56 \mapsto \bar246$, $246 \mapsto 46$; and 46 subsumes 456, $46\bar7\bar9$, $\bar246$. Similarly, $\bar2\bar567 \mapsto \bar267$, $\bar4\bar579 \mapsto \bar459$, $\bar2\bar347 \mapsto \bar2\bar37$, $\bar34\bar57 \mapsto \bar3\bar57$, $\bar3\bar5\bar7 \mapsto \bar3\bar5$; and $\bar3\bar5$ subsumes $\bar34\bar5$, $\bar3\bar57$. Then $2\bar45\bar6 \mapsto 2\bar4\bar6$, $\bar2\bar4\bar6 \mapsto \bar4\bar6$; and $\bar4\bar6$ subsumes $\bar45\bar6$, $2\bar4\bar6$, $\bar4\bar679$. Also $25\bar6\bar7 \mapsto 2\bar6\bar7$, $4\bar57\bar9 \mapsto 4\bar5\bar9$, $25\bar7\bar9 \mapsto 2\bar7\bar9$.

Round 2 of variable elimination first gets rid of 4, replacing six clauses by just four using exercise 369: $23\bar6$, $\bar2\bar36$, 569, $\bar5\bar6\bar9$. Then variable 3 goes away; ten clauses become eight, again via exercise 369: $2\bar5\bar6$, $\bar256$, $2\bar5\bar7$, $\bar257$, $2\bar5\bar9$, $\bar259$, $56\bar9$, $\bar569$. And the ten clauses that now contain 2 or $\bar2$ resolve into just four: $56\bar7\bar9$, $5\bar679$, $\bar567\bar9$, $\bar5\bar679$.

After eliminating 7 and 9, only four clauses remain, namely 56, $5\bar6$, $\bar56$, $\bar5\bar6$; and they quickly produce a contradiction.

372. (This problem is surprisingly difficult.) Are the clauses $\{\bar1\bar5, \bar1\bar6, \bar2\bar5, \bar2\bar6, \bar3\bar7, \bar3\bar8, \bar4\bar7, \bar4\bar8, 123, 124, 134, 234, 567, 568, 578, 678\}$ as "small" as possible?

373. Using the notation of (102), elimination of $x_{1m}, x_{2m}, \ldots, x_{mm}$ produces new clauses M'_{imk} for $1 \le i, k < m$ as well as $M_{m(m-1)}$. Then elimination of $x_{m(m-1)}$ gives $(M_{i(m-1)} \vee M_{m(m-2)})$ for $1 \le i < m$. This clause self-subsumes to $M_{i(m-1)}$, using $M'_{im1}, \ldots, M'_{im(m-2)}$. And $M_{i(m-1)}$ subsumes each M'_{imk}, so we've reduced m to $m-1$.

374. As in (57), variables are numbered 1 to n, and literals from 2 to $2n+1$. But we will now number the clauses from $2n+2$ to $m+2n+1$. The literals of clauses will be stored in cells, somewhat as in Algorithm A, but with additional links as in the exact cover algorithms of Section 7.2.2.1: Each cell p contains not only a literal `L(`p`)`, a clause number `C(`p`)`, and forward/backward pointers `F(`p`)` and `B(`p`)` to other cells with the same literal, but also left/right pointers `S(`p`)` and `D(`p`)` to other cells in the same *clause*. (Think "sinister" and "dexter.") Cells 0 and 1 are reserved for special use; cell l, for $2 \le l < 2n+2$, serves as the head of the doubly linked list of cells that contain the literal l; cell c, for $2n+2 \le c < m+2n+2$, serves as the head of the doubly linked list of cells that contain the elements of clause c; and cell p, for $m+2n+2 \le p < M$, either is available for future use or holds literal and clause data for a currently active clause.

Free cells are accessed via a global pointer `AVAIL`. To get a new $p \Leftarrow$ `AVAIL` when `AVAIL` $\ne 0$, we set $p \leftarrow$ `AVAIL`, `AVAIL` $\leftarrow$ `S(AVAIL)`; but if `AVAIL` $= 0$, we set $p \leftarrow M$ and $M \leftarrow M+1$ (assuming that M never gets too large). To free one or more cells from p' to p'' that are linked together via left links, we set `S(`p'`)` $\leftarrow$ `AVAIL` and `AVAIL` $\leftarrow p''$.

hyperresolution
substitution
DNF
Two-level circuit minimization
irredundant
self-subsumptions
self-subsumes
exact cover
doubly linked
AVAIL

The number of active clauses containing literal l, TALLY(l), can therefore be computed as follows: Set $t \leftarrow 0$, $p \leftarrow$ F(l); while not $lit(p)$, set $t \leftarrow t+1$ and $p \leftarrow$ F(p); set TALLY(l) $\leftarrow t$; here '$lit(p)$' stands for '$p < 2n+2$'. The number of literals in clause c, SIZE(c), can be computed by a similar loop, using '$cls(p)$' to stand for '$p < m+2n+2$': Set $t \leftarrow 0$, $p \leftarrow$ S(c); while not $cls(p)$, set $t \leftarrow t+1$ and $p \leftarrow$ S(p); set SIZE(c) $\leftarrow t$. After initialization, the TALLY and SIZE statistics can be updated dynamically as local changes are made. (TALLY(l) and SIZE(c) can be maintained in L(l) and C(c).)

To facilitate resolution, the literals of each clause are required to increase from left to right; in other words, we must have L(p) $<$ L(q) whenever $p =$ S(q) and $q =$ D(p), unless $cls(p)$ or $cls(q)$. But the clauses within literal lists need not appear in any particular order. We might even have C(F(p)) $>$ C(q) but C(F(p')) $<$ C(q'), when C(p) $=$ C(p') and C(q) $=$ C(q').

To facilitate subsumption, each literal l is assigned a 64-bit *signature* SIG(l) $=$ $(1 \ll U_1) \mid (1 \ll U_2)$, where U_1 and U_2 are independently random 6-bit numbers. Then each clause c is assigned a signature that is the bitwise OR of the signatures of its literals: Set $t \leftarrow 0$, $p \leftarrow$ S(c); while not $cls(p)$, set $t \leftarrow t \mid$ SIG(L(p)) and $p \leftarrow$ S(p); set SIG(c) $\leftarrow t$. (See the discussion of Bloom's superimposed coding in Section 6.5.)

(a) To resolve c with c', where c contains l and c' contains $\bar{l}$, we essentially want to do a list merge. Set $p \leftarrow 1$, $q \leftarrow$ S(c), $u \leftarrow$ L(q), $q' \leftarrow$ S(c'), $u' \leftarrow$ L(q'), and do the following while $u + u' > 0$: If $u = u'$, copy(u) and bump(q, q'); if $u = \bar{u}' = l$, bump(q, q'); if $u = \bar{u}' \neq l$, terminate unsuccessfully; otherwise if $u > u'$, copy(u) and bump(q); otherwise copy(u') and bump(q'). Here 'copy(u)' means 'set $p' \leftarrow p$, $p \Leftarrow$ AVAIL, S(p') $\leftarrow p$, L(p) $\leftarrow u$'; 'bump(q)' means 'set $q \leftarrow$ S(q); if $cls(q)$ set $u \leftarrow 0$, otherwise set $u \leftarrow$ L(q)'; 'bump(q')' is similar, but it uses q' and u'; and 'bump(q, q')' means 'bump(q) and bump(q')'. Unsuccessful termination occurs when clauses c and c' resolve to a tautology; we set $p \leftarrow 0$, after first returning cells p through S(1) to free storage if $p \neq 1$. Successful termination with $u = u' = 0$ means that the resolved clause consists of the literals in cells from p through S(1), linked only via S pointers.

(b) Find a literal l with minimum TALLY(l). Set $p \leftarrow$ F(l), and do the following while not $lit(p)$: Set $c' \leftarrow$ C(p); if $c' \neq c$ and $\sim$SIG(c') & SIG(c) $= 0$ and SIZE(c') $\geq$ SIZE(c), do a detailed subsumption test; then set $p \leftarrow$ F(p). The detailed test begins with $q \leftarrow$ S(c), $u \leftarrow$ L(q), $q' \leftarrow$ S(c'), $u' \leftarrow$ L(q'), and does the following steps while $u' \geq u > 0$: bump(q') while $u' > u$; then bump(q, q') if $u' = u$. When the loop terminates, c subsumes c' if and only if $u \leq u'$.

(c) Use (b) but with (SIG(c) & $\sim$SIG(l)) in place of SIG(c). Also modify the detailed test, by inserting 'if $u = l$ then $u \leftarrow \bar{l}$' just after each occurrence of '$u \leftarrow$ L(q)'.

[The algorithm in (b) was introduced by A. Biere, *LNCS* **3542** (2005), 59–70, §4.2. "False hits," in which the detailed test is performed but no actual (self-)subsumption is detected, tend to occur less than 1% of the time in practice.]

375. Let each literal l have another field STAMP(l), initially zero; and let s be a global "time stamp" that is initially zero. To make the test, set $s \leftarrow s+1$ and $\sigma \leftarrow 0$; then set STAMP(u) $\leftarrow s$ and $\sigma \leftarrow \sigma \mid$ SIG(u) for all u such that $(\bar{l}\bar{u})$ is a clause. If $\sigma \neq 0$, set $\sigma \leftarrow \sigma \mid$ SIG(l) and run through all clauses c that contain l, doing the following: If SIG(c) & $\sim\sigma = 0$, and if each of c's literals $u \neq l$ has STAMP(u) $= s$, exit with $C_1 = c$ and $r =$ SIZE(c) $- 1$. If C_1 has thereby been found, set $s \leftarrow s+1$ and STAMP($\bar{u}$) $\leftarrow s$ for all $u \neq l$ in c. Then a clause $(\bar{l} \vee \beta_j)$ implicitly has $j \leq r$ in the notation of exercise 369 if and only if β_j is a single literal u with STAMP(u) $= s$.

Given a variable x, test the condition first for $l = x$; if that fails, try $l = \bar{x}$.

signature
bitwise OR
Bloom
list merge
tautology
Biere
False hits
STAMP(l)
time stamp

376. Highest priority is given to the common operations of unit conditioning and pure literal elimination, which are "low-hanging fruit." Give each variable x two new fields, STATE(x) and LINK(x). A "to-do stack," containing all such easy pickings, begins at TODO and follows LINKs until reaching Λ. Variable x is on this stack only if STATE(x) is nonzero; the nonzero states are called FF (forced false), FT (forced true), EQ (eliminated quietly), and ER (eliminated by resolution).

Whenever a unit clause (l) is detected, with STATE($|l|$) $= 0$, we set STATE($|l|$) $\leftarrow$ (l & 1? FF: FT), LINK($|l|$) $\leftarrow$ TODO, and TODO $\leftarrow |l|$. But if STATE($|l|$) $=$ (l & 1? FT: FF), we terminate, because the clauses are unsatisfiable.

Whenever a literal with TALLY(l) $= 0$ is detected, we do the same thing if STATE($|l|$) $= 0$. But if STATE($|l|$) $=$ (l & 1? FT: FF), we simply set STATE($|l|$) $\leftarrow$ EQ instead of terminating.

To clear the to-do stack, we do the following while TODO $\neq \Lambda$: Set $x \leftarrow$ TODO and TODO $\leftarrow$ LINK(x); if STATE(x) $=$ EQ, do nothing (no erp rule is needed to eliminate x); otherwise set $l \leftarrow$ (STATE(x) $=$ FT? x: $\bar{x}$), output the erp rule $l \leftarrow 1$, and use the doubly linked lists to delete all clauses containing l and to delete $\bar{l}$ from all clauses. (Those deletions update TALLY and SIZE fields, so they often contribute new entries to the to-do stack. Notice that if clause c loses a literal, we must recompute SIG(c). If clause c disappears, we set SIZE(c) $\leftarrow 0$, and never use c again.)

Subsumption and strengthening are next in line. We give each clause c a new field LINK(c), which is nonzero if and only if c appears on the "exploitation stack." That stack begins at EXP and follows LINKs until reaching the nonzero sentinel value Λ'. All clauses are initially placed on the exploitation stack. Afterwards, whenever a literal $\bar{l}$ is deleted from a clause c, either during unit conditioning or self-subsumption, we test if LINK(c) $= 0$; if so, we put c back on the stack by setting LINK(c) $\leftarrow$ EXP and EXP $\leftarrow c$.

To clear the subsumption stack, we first clear the to-do stack. Then, while EXP $\neq$ Λ', we set $c \leftarrow$ EXP, EXP $\leftarrow$ LINK(c), and do the following if SIZE(c) $\neq 0$: Remove clauses subsumed by c; clear the to-do stack; and if SIZE(c) is still nonzero, strengthen clauses that c can improve, clear the to-do stack, and set TIME(c) $\leftarrow T$ (see below).

All of this takes place before we even think about the elimination of variables. But rounds of variable elimination form the "outer level" of computation. Each variable x has yet another field, STABLE(x), which is nonzero if and only if we need not attempt to eliminate x. This field is initially zero, but set nonzero when STATE(x) $\leftarrow$ EQ or when an erp rule for x or $\bar{x}$ is output. It is reset to zero whenever a variable is later "touched," namely when x or $\bar{x}$ appears in a deleted or self-subsumed clause. (In particular, every variable that appears in a new clause produced by resolution will be touched, because it will appear in at least one of the clauses that were replaced by new ones.)

If a round has failed to eliminate any variables, or if it has eliminated them all, we're done. But otherwise there's still work to do, because the new clauses can often be subsumed or strengthened. (Indeed, some of them might actually be duplicates.) Hence two *more* fields are introduced: TIME(l) for each literal and TIME(c) for each clause, initially zero. Let T be the number of the current elimination round. We set TIME(l) $\leftarrow T$ for all literals l in all clauses that are replaced by resolution, and TIME(c) $\leftarrow T$ is also set appropriately as mentioned above.

Introduce yet another field, EXTRA(c), initially zero. It is reset to zero whenever TIME(c) $\leftarrow T$, and set to 1 whenever c is replaced by a new clause. For every literal l such that STATE($|l|$) $= 0$ and TIME(l) $= T$ at the end of round T, set EXTRA(c) $\leftarrow$ EXTRA(c) $+ 4$ for all clauses c that contain l, and EXTRA(c) $\leftarrow$ EXTRA(c) $\mid 2$ for all clauses c that contain $\bar{l}$. Then run through all clauses c for which SIZE(c) > 0 and

unit conditioning
pure literal elimination
to-do stack
erp rule
doubly linked lists
Subsumption
strengthening
exploitation stack
sentinel value
touched

TIME$(c) < T$. If SIZE$(c) =$ EXTRA$(c) \gg 2$, remove clauses subsumed by c and clear the exploitation stack. Also, if EXTRA(c) & $3 \neq 0$, we may be able to use c to strengthen other clauses — unless EXTRA(c) & $1 = 0$ and EXTRA$(c) \gg 2 <$ SIZE$(c) - 1$. Self-subsumption using l need not be attempted when EXTRA(c) & $1 = 0$ unless TIME$(\bar{l}) = T$ and EXTRA$(c) \gg 2 =$ SIZE$(c) - [$TIME$(l) = T]$. Finally, reset EXTRA(c) to zero (even if TIME$(c) = T$). [See Niklas Eén and Armin Biere, *LNCS* **3569** (2005), 61–75.]

377. Each vertex v of G corresponds to variables v_1, v_2, v_3 in F; each edge $u — v$ corresponds to clauses $(\bar{u}_1 \vee v_2)$, $(\bar{u}_2 \vee v_3)$, $(\bar{u}_3 \vee \bar{v}_1)$, $(u_2 \vee \bar{v}_1)$, $(u_3 \vee \bar{v}_2)$, $(\bar{u}_1 \vee \bar{v}_3)$. The longest paths in the dependency digraph for F have the form $t_1 \to u_2 \to v_3 \to \bar{w}_1$ or $t_1 \to \bar{u}_3 \to \bar{v}_2 \to \bar{w}_1$, where $t — u — v — w$ is a walk in G.

[A similar method reduces the question of finding an oriented cycle of length r in a given digraph to the question of finding a failed literal in some dependency digraph. The cycle detection problem has a long history; see N. Alon, R. Yuster, and U. Zwick, *Algorithmica* **17** (1997), 209–223. So any surprisingly fast algorithm to decide whether or not failed literals exist — that is, faster than $n^{2\omega/(\omega+1)}$ when $m = O(n)$ and matrix multiplication takes $O(n^\omega)$ — would lead to surprisingly fast algorithms for other problems.]

378. The erp rule $l \leftarrow l \vee (\bar{l}_1 \wedge \cdots \wedge \bar{l}_q)$ will change any solution of $F \setminus C$ into a solution of F. [See M. Järvisalo, A. Biere, and M. Heule, *LNCS* **6015** (2010), 129–144.]

(In practice it's sometimes possible to remove tens of thousands of blocked clauses. For example, all of the exclusion clauses (17) in the coloring problem are blocked, as are many of the clauses that arise in fault testing. Yet the author has yet to see a *single* example where blocked clause elimination is actually helpful in combination with transformations 1–4, which are already quite powerful by themselves.)

379. (Solution by O. Kullmann.) In general, *any* set F of clauses can be replaced by another set F', whenever there's a variable x such that the elimination of x from F yields exactly the same clauses as the elimination of x from F'. In this case the elimination of a has this property. The erp rule $a \leftarrow a \vee (\bar{b} \wedge \bar{c} \wedge d)$ is necessary and sufficient.

380. (a) Reverse self-subsumption weakens it to $(a \vee b \vee c \vee d)$, then to $(a \vee b \vee c \vee d \vee e)$, which is subsumed by $(a \vee d \vee e)$. [In general one can show that reverse self-subsumption from C leads to a subsumed clause if and only if C is certifiable from the other clauses.]

(b) Again we weaken to $(a \vee b \vee c \vee d \vee e)$; but now we find this blocked by c.

(c) No erp rule is needed in (a), but we need $c \leftarrow c \vee (\bar{a} \wedge \bar{b})$ in (b). [Heule, Järvisalo, and Biere, *LNCS* **6397** (2010), 357–371, call this "asymmetric elimination."]

381. By symmetry, we'll remove the final clause. (Without it, the given clauses state that $x_1 \le x_2 \le \cdots \le x_n$; with it, they state that all variables are equal.) Assume more generally that, for $1 \le j < n$, every clause other than $(\bar{x}_j \vee x_{j+1})$ that contains $\bar{x}_j$ also contains either x_n or $\bar{x}_i$ for some $i < j$. For $1 \le j < n-1$ we can then weaken $(x_1 \vee \cdots \vee x_j \vee \bar{x}_n)$ to $(x_1 \vee \cdots \vee x_{j+1} \vee \bar{x}_n)$. Finally, $(x_1 \vee \cdots \vee x_{n-1} \vee \bar{x}_n)$ can be eliminated because it is blocked by x_{n-1}.

Although we've eliminated only one clause, $n-1$ erp rules are actually needed to undo the process: $x_1 \leftarrow x_1 \vee x_n$; $x_2 \leftarrow x_2 \vee (\bar{x}_1 \wedge x_n)$; $x_3 \leftarrow x_3 \vee (\bar{x}_1 \wedge \bar{x}_2 \wedge x_n)$; $\ldots$; $x_{n-1} \leftarrow x_{n-1} \vee (\bar{x}_1 \wedge \cdots \wedge \bar{x}_{n-2} \wedge x_n)$. (Those rules, applied in reverse order, can however be simplified to $x_j \leftarrow x_j \vee x_n$ for $1 \le j < n$, because $x_1 \le \cdots \le x_n$ in any solution.)

[See Heule, Järvisalo, Biere, *EasyChair Proc. in Computing* **13** (2013), 41–46.]

382. See M. J. H. Heule, M. Järvisalo, and A. Biere, *LNCS* **6695** (2011), 201–215.

383. (a) In a learning step, let $\Phi' = \Phi$ and $\Psi' = \Psi \cup C$. In a forgetting step, let $\Phi' = \Phi$ and $\Psi = \Psi' \cup C$. In a hardening step, let $\Phi' = \Phi \cup C$ and $\Psi = \Psi' \cup C$. In a softening

Eén
Biere
dependency digraph
walk
oriented cycle
cycle detection problem
Alon
Yuster
Zwick
matrix multiplication
Järvisalo
Biere
exclusion clauses
coloring
fault testing
author
Kullmann
elimination of x
certifiable
blocked
Heule
Järvisalo
Biere
asymmetric elimination
blocked
Heule
Järvisalo
Biere
Heule
Järvisalo
Biere

invariant
Järvisalo
Heule
Biere
unit conditioning
resolution certifiable
Blocked clauses
RAT, see resolution certifiable clauses

step, let $\Phi = \Phi' \cup C$ and $\Psi' = \Psi \cup C$. In all four cases it is easy to verify that $(\text{sat}(\Phi) \iff \text{sat}(\Phi \cup \Psi))$ implies $(\text{sat}(\Phi) \iff \text{sat}(\Phi') \iff \text{sat}(\Phi' \cup \Psi'))$, where sat$(G)$ means "G is satisfiable," because $\text{sat}(G \cup G') \implies \text{sat}(G)$. Thus the assertions are invariant.

(b) Each erp rule allows us to go one step backward, until reaching F.

(c) The first (softening) step is fine, because both $\Phi = (x)$ and $\Phi \setminus (x) = 1$ are satisfiable, and because the erp rule unconditionally makes x true. But the second (learning) step is flawed, because $\text{sat}(\Phi \cup \Psi)$ does not imply $\text{sat}(\Phi \cup \Psi \cup C)$ when $\Phi \cup \Psi = (x)$ and $C = (\bar{x})$. (This example explains why the criterion for learning is not simply '$\text{sat}(\Phi) \implies \text{sat}(\Phi \cup C)$' as it essentially is for softening.)

(d) Yes, because C is also certifiable for $\Phi \cup \Psi$.

(e) Yes, after softening it. No erp rule is needed, because $\Phi \setminus C \vdash C$.

(f) A soft clause can be discarded whether or not it is subsumed. To discard a hard clause that is subsumed by a soft clause, first harden the soft one. To discard a hard C that is subsumed by a hard C', weaken C and then discard it. (The weakening step is clearly permissible, and no erp rule is needed.)

(g) If C contains $\bar{x}$ and C' contains x and $C \setminus \bar{x} \subseteq C' \setminus x$, we can learn the soft clause $C \diamond C' = C' \setminus x$, then use it to subsume C' as in (f).

(h) Forget all soft clauses that contain x or $\bar{x}$. Then let $C_1, \ldots, C_p$ be the hard clauses containing x, and $C'_1, \ldots, C'_q$ those containing $\bar{x}$. Learn all the (soft) clauses $C_i \diamond C'_j$, and harden them, noting that they don't involve x. Weaken each C_i, with erp rule $x \leftarrow x \vee \overline{C}_i$, and forget it; also weaken and forget each C'_j, with erp rule $x \leftarrow x \wedge C'_j$. (One can show that either of the erp rules in (161) would also suffice.)

(i) Whenever $\Phi \cup \Psi$ is satisfiable, so is $\Phi \cup \Psi \cup \{(x \vee z), (y \vee z), (\bar{x} \vee \bar{y} \vee \bar{z})\}$, because we can always set $z \leftarrow \bar{x} \vee \bar{y}$.

[*Reference:* M. Järvisalo, M. Heule, and A. Biere, *LNCS* **7364** (2012), 355–370. Notice that, by exercise 368, parts (f) and (h) justify the use of unit conditioning.]

384. Whenever we have a solution to $\Phi \setminus C$ that falsifies C, we will show that Φ is satisfied by making l true; hence softening C is permissible, with erp rule $l \leftarrow l \vee \overline{C}$.

To prove that claim, notice that a problem could arise only in a hard clause C' that contains $\bar{l}$. But if all other literals of C' are false in the given solution, then all literals of $C \diamond C'$ are false, contradicting the assumption that $(\Phi \setminus C) \wedge \overline{C \diamond C'} \vdash_1 \epsilon$.

(Such clauses C are "resolution certifiable" with respect to $\Phi \setminus C$. Blocked clauses are a very special case. Similarly, we can safely learn any clause that is resolution certifiable with respect to $\Phi \cup \Psi$.)

385. (a) True, because $\overline{C} \wedge l \vdash_1 \epsilon$.

(b) $\bar{1}$ is implied, not certifiable; $\bar{1}2$ is certifiable, not absorbed; $\bar{1}23$ is absorbed.

(c, d) If C is any clause and l is any literal, then $F \wedge \overline{C} \vdash_1 l$ implies $F' \wedge \overline{C} \vdash_1 l$, because unit propagation in F carries over to unit propagation in F'.

386. (a) The trail contained exactly score(F, C, l) literals when decision $\bar{l}$ was made at level d. The clause learned from the ensuing conflict causes at least one new literal to be implied at level $d' < d$.

(b) The score can't decrease when F grows.

(c) Each $l \in C$ needs at most n helpful rounds to make score$(F, C, l) = \infty$.

(d) Suppose, for example, $F = (a \vee \bar{d}) \wedge (a \vee b \vee e \vee l) \wedge (\bar{a} \vee c) \wedge (\bar{b}) \wedge (c \vee d \vee \bar{e} \vee l)$ and $C = (a \vee b \vee c \vee d \vee l)$. The helpful sequences of decisions are $(\bar{a}, \bar{c}, \bar{l})$, $(\bar{c}, \bar{d}, \bar{l})$, $(\bar{d}, \bar{a}, \bar{c}, \bar{l})$, $(\bar{d}, \bar{c}, \bar{l})$, and they occur with probabilities $\frac{1}{10}\frac{1}{6}\frac{1}{4}$, $\frac{1}{10}\frac{1}{6}\frac{1}{4}$, $\frac{1}{10}\frac{1}{8}\frac{1}{6}\frac{1}{4}$, $\frac{1}{10}\frac{1}{8}\frac{1}{4}$.

In general if a decision is to be made and j elements of $\overline{C}$ are not yet in the trail, the probability that a suitable decision will be made at random is at least

$$f(n,j)=\min\Bigl(\frac{j-1}{2n}f(n-1,j-1),\ \frac{j-2}{2(n-1)}f(n-2,j-2),\ \ldots,\ \frac{1}{2(n-j+2)}f(n-j+1,1),\ \frac{1}{2(n-j+1)}\Bigr)=\frac{(j-1)!}{2^j n^{\underline{j}}}.$$

(e) The waiting time to absorb each clause C_i is a geometric distribution whose mean is $\le 4n^{|C_i|}$, repeated at most $|C_i|n$ times.

References: K. Pipatsrisawat and A. Darwiche, *Artif. Intell.* **175** (2011), 512–525; A. Atserias, J. K. Fichte, and M. Thurley, *J. Artif. Intell. Research* **40** (2011), 353–373.

387. We may assume that G and G' have no isolated vertices. Letting variable vv' mean that v corresponds to v', we need the clauses $(\overline{uv'}\vee\overline{vv'})$ for $u<v$ and $(\overline{vu'}\vee\overline{vv'})$ for $u'<v'$. Also, for each $u<v$ with $u \mathbin{—} v$ in G, we introduce auxiliary variables $uu'vv'$ for each edge $u' \mathbin{—} v'$ in G', with clauses $(\overline{uu'vv'}\vee uu')\wedge(\overline{uu'vv'}\vee vv')\wedge(\bigvee\{uu'vv' \mid u' \mathbin{—} v' \text{ in } G'\})$. The variables vv' and $uu'vv'$ can be restricted to cases where $\mathrm{degree}(u)\le\mathrm{degree}(u')$ and $\mathrm{degree}(v)\le\mathrm{degree}(v')$.

388. (a) Can the complete graph K_k be embedded in G? (b) Can G be embedded in the complete k-partite graph $K_{n,\ldots,n}$, where G has n vertices? (c) Can the cycle C_n be embedded in G?

389. This is a graph embedding problem, with G' the 4×4 (king $\cup$ knight) graph and with G defined by edges `T` — `H`, `H` — `E`, ..., `N` — `G`. The adjacent `M`s can be avoided by changing '`PROGRAMMING`' to either '`PROGRAMXING`' or '`PROGRAXMING`'.

Algorithm C needs less than 10 megamems to find the first solution below. Furthermore, if the blank space can also be moved, the algorithm will rather quickly also find solutions with just five knight moves (the minimum), or 17 of them (the max):

U	P	C	F
M	M	O	␣
I	T	R	A
N	G	E	H

M	M	I	N
A	P	O	G
H	R	␣	F
U	T	E	C

H	N	U	F
E	M	O	I
G	T	␣	P
A	R	M	C

390. Let $d(u,v)$ be the distance between vertices u and v. Then $d(v,v)=0$ and

$$d(u,v)\le j+1 \iff d(u,v)\le j \text{ or } d(u,w)\le j \text{ for some } w\in N(v)=\{w\mid w \mathbin{—} v\}. \quad (*)$$

In parts (a), (d), we introduce variables v_j for each vertex v and $0\le j\le k$. In part (c) we do this for $0\le j<n$. But parts (b), (e), (f) use just n variables, $\{v\mid v\in V\}$.

(a) Clauses $(s_0)\wedge\bigvee_{v\in V\setminus s}(\bar v_0)\wedge\bigvee_{v\in V}(\bar v_{j+1}\vee v_j\vee\bigvee_{w — v} w_j)$ are satisfied only if $v_j\le[d(s,v)\le j]$; hence the additional clause (t_k) is also satisfied only if $d(s,t)\le k$. Conversely, if $d(s,t)\le k$, all clauses are satisfied by setting $v_j\leftarrow[d(s,v)\le j]$.

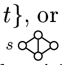
(b) There's a path from s to t if and only if there's a subset $H\subseteq V$ such that $s\in H$, $t\in H$, and every other vertex of the induced graph $G\,|\,H$ has degree 0 or 2. [The vertices on a shortest path from s to t yield one such H. Conversely, given H, we can find vertices $v_j\in H$ such that $s=v_0 — v_1 — \cdots — v_k=t$.]

We can represent that criterion via clauses on the binary variables $v=[v\in H]$ by asserting $(s)\wedge(t)$, together with clauses to ensure that $\Sigma(s)=\Sigma(t)=1$, and that $\Sigma(v)\in\{0,2\}$ for all $v\in H\setminus\{s,t\}$, where $\Sigma(v)=\sum_{w\in N(v)}w$ is the degree of v in $G\,|\,H$. The number of such clauses for each v is at most $6|N(v)|$, because we can append $\bar v$

geometric distribution
Pipatsrisawat
Darwiche
Atserias
Fichte
Thurley
isolated vertices
auxiliary variables
complete graph
complete k-partite graph
cycle
graph embedding
distance $d(u,v)$ in a graph
induced graph
shortest path

digraphs
strong connectivity
Heule
renamed
Horn clauses
Van Gelder
Loyd
Dudeney
Grabarchuks'
odd-even transposition sort

to each clause of (18) and (19) when $r = 2$, and $|N(v)|$ additional clauses will rule out $\Sigma(v) < 2$. Altogether there are $O(m)$ clauses, because $\sum_{v\in V} |N(v)| = 2m$.

[Similar but simpler alternatives, such as (i) to require $\Sigma(v) \in \{0, 2\}$ for all $v \in V\setminus\{s,t\}$, or (ii) to require $\Sigma(v) \geq 2$ for all $v \in H\setminus\{s,t\}$, do *not* work: Counterexamples are (i)  and (ii) . Another solution, more cumbersome, associates a Boolean variable with each *edge* of G.]

(c) Let s be any vertex; use (a), plus (v_{n-1}) for all $v \in V \setminus s$.

(d) Clauses $(s_0)\wedge\bigvee_{j=0}^{k-1}\bigvee_{v\in V}\bigvee_{w\in N(v)}(\bar v_k\vee w_{k+1})$ are satisfied only if we have $v_j \geq [d(s,v)\leq j]$; hence the additional clause $(\bar t_k)$ cannot also be satisfied when $d(s,t) \leq k$. Conversely, if $d(s,t) > k$ we can set $v_j \leftarrow [d(s,v)\leq j]$.

(e) $(s) \wedge \bigl(\bigvee_{v\in V}\bigvee_{w\in N(v)}(\bar v \vee w)\bigr) \wedge (\bar t)$.

(f) Letting s be any vertex, use $(s) \wedge \bigl(\bigvee_{v\in V}\bigvee_{w\in N(v)}(\bar v \vee w)\bigr) \wedge \bigl(\bigvee_{v\in V\setminus s}\bar v\bigr)$.

[Similar constructions work with digraphs and strong connectivity. Parts (d)–(f) of this exercise were suggested by Marijn Heule. Notice that parts (a) and (c)–(f) construct renamed Horn clauses, which work very efficiently (see exercise 444).]

391. (a) Let $d - 1 = (q_{l-1}\ldots q_0)_2$. To ensure that $(x_{l-1}\ldots x_0)_2 < d$ we need the clauses $(\bar x_i \vee \bigvee\{\bar x_j \mid j > i,\ q_j = 1\})$ whenever $q_i = 0$. The same holds for y.

To enforce $x \neq y$, introduce the clause $(a_{l-1} \vee \cdots \vee a_0)$ in auxiliary variables $a_{l-1}\ldots a_0$, together with $(\bar a_j \vee x_j \vee y_j) \wedge (\bar a_j \vee \bar x_j \vee \bar y_j)$ for $0 \leq j < l$ (see (172)).

(b) Now $x \neq y$ is enforced via clauses of length $2l$, which state that we don't have $x = y = k$ for $0 \leq k < d$. For example, the appropriate clause when $l = 3$ and $k = 5$ is $(\bar x_2 \vee \bar y_2 \vee x_1 \vee y_1 \vee \bar x_0 \vee \bar y_0)$.

(c) Use the clauses of (b) for $0 \leq k < 2d - 2^l$, plus clauses of length $2l - 2$ for $d \leq k < 2^l$ stating that we don't have $(x_{l-1}\ldots x_1)_2 = (y_{l-1}\ldots y_1)_2 = k$. (The encodings in (b) and (c) are identical when $d = 2^l$.)

[See A. Van Gelder, *Discrete Applied Mathematics* **156** (2008), 230–243.]

392. (a) [Puzzle (*ii*) was introduced by Sam Loyd in the *Boston Herald*, 13 November 1904; page 27 of his *Cyclopedia* (1914) states that he'd created a puzzle like (*i*) at age 9! Puzzle (*iv*) is by H. E. Dudeney, *Strand* **42** (1911), 108, slightly modified. Puzzle (*iii*) is from the Grabarchuks' *Big, Big, Big Book of Brainteasers* (2011), #196; puzzle (*v*) was designed by Serhiy A. Grabarchuk in 2015.]

A	A	A	A	A
A	B	B	B	B
A	A	A	A	A
C	C	C	C	A
A	A	A	A	A

(*i*)

A	A	A	A	D	D	D	D
A	D	D	D	D	E	E	D
A	A	A	A	A	A	E	D
C	C	C	C	C	A	E	D
C	A	A	A	C	A	E	D
C	A	B	A	A	A	E	B
C	A	B	B	B	E	E	B
C	A	A	A	B	B	B	B

(*ii*)

A	A	A	B	B	B	B	B
A	C	A	A	A	C	C	B
A	C	C	C	A	C	B	B
A	A	D	C	C	C	B	E
F	A	D	D	D	E	B	E
F	A	A	A	D	E	B	E
F	D	D	D	D	E	B	E
F	F	F	F	F	E	E	E

(*iii*)

G	G	G	G	G	H	A	A	A	A	A	A	A	C	C	C	C	C
G	F	F	F	G	H	A	C	C	C	C	C	C	C	E	E	E	C
G	F	B	F	G	H	A	C	E	E	E	E	E	E	E	D	E	C
G	F	B	F	G	H	A	A	E	A	A	A	A	A	A	D	E	E
G	F	B	F	G	H	H	A	A	A	G	H	H	H	A	D	D	D
G	F	B	F	G	G	H	H	H	H	H	H	G	H	A	A	A	D
F	F	B	F	F	G	G	G	G	G	G	G	G	H	H	H	A	D
F	B	B	J	F	F	F	F	F	F	F	F	F	F	F	H	A	D
B	B	J	J		H	H	H	H	H	H	H	H	H	H	H	A	D
B	I	I	I	I	H	I	I	I	I	I	I	A	A	A	A	A	D
B	B	B		I	I	I	D	D	D	D	D	D	D	D	D	D	D

(*iv*)

E	E	E	E	E	A	A	A	A	A	A	A	A	A	A	A	A	A
E	D	D	D	E	E	C	C	C	C	C	C	C	C	C	C	C	A
E	D	B	D	C	E	C	B	B	B	B	B	B	B	B	B	C	A
E	D	B	D	C	E	C	B	F	F	F	F	F	F	F	B	C	A
E	D	B	D	C	E	C	B	F	D	D	D	D	D	F	B	C	A
E	D	B	B	C	E	C	B	F	D	E	E	E	D	F	B	C	A
E	D	F	B	C	C	C	B	F	D	E	A	E	D	F	B	C	A
E	D	F	B	B	B	B	B	F	D	E	A	C	B	F	B	C	A
E	D	F	F	F	F	F	F	F	D	E	A	C	B	B	B	C	A
E	D	D	D	D	D	D	D	D	D	E	A	C	C	C	C	C	A
E	E	E	E	E	E	E	E	E	E	E	A	A	A	A	A	A	A

(*v*)

(b) [Puzzle (*vi*) is an instance of the odd-even transposition sort, exercise 5.3.4–37. Eight order-reversing connections would be impossible with only eight columns, instead of the nine in (*vii*), because the permutation has too many inversions.]

A	B	B	D	D	F	F	H	H
B	A	D	B	F	D	H	F	G
C	D	A	F	B	H	D	G	F
D	C	F	A	H	B	G	D	E
E	F	C	H	A	G	B	E	D
F	E	H	C	G	A	E	B	C
G	H	E	G	C	E	A	C	B
H	G	G	E	E	C	C	A	A

(*vi*)

A	B	B	D	D	F		
B	A	D	B	F	D	F	G
C	D	A	F	B	D	G	F
D	C	F	A	B	G	D	E
E	F	C	A	G	B	E	D
F	E	C	G	A	E	B	C
G	E	G	C	E	A	C	B
	G	E	E	C	C	A	A

(*vii*)

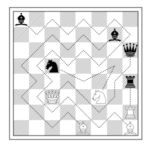
A	B	B	C	C	D	D	A
D	A	C	B	D	C	A	B
D	C	A	B	D	A	C	B
C	D	B	A	A	D	B	C
C	B	D	B	D	A	D	B
B	C	C	D	B	A	B	D
B	A	D	C	A	B	A	D
A	D	A	A	C		B	A

(*viii*)

(c) Let $d_j = \sum_{i=1}^{j}(|T_i|-1)$ and $d = d_t$. We introduce variables v_i for $1 \le i \le d$, and the following clauses for $1 \le j \le t$ and $d_{j-1} < i \le d_j$: $(\bar v_{i'} \vee \bar v_i)$ for $1 \le i' \le d_{j-1}$; the clauses of answer 390(b) on variables v_i, where s is the $(i-d_{j-1})$th element of T_j and t is the last element. These clauses ensure that the sets $V_j = \{v \mid v_{d_{j-1}+1} \vee \cdots \vee v_{d_j}\}$ are disjoint, and that V_j contains a connected component $S_j \supseteq T_j$.

We also assert $(\bar v_i)$ for $1 \le i \le d$, whenever T_j is a singleton set $\{v\}$.

[For the more general "Steiner tree packing" problem, see M. Grötschel, A. Martin, and R. Weismantel, *Math. Programming* **78** (1997), 265–281.]



393. A construction somewhat like that of answer 392(c) can be used with five different 8×8 graphs, one for the moves of each white-black pair S_j. But we need to keep track of the edges used, not vertices, in order to prohibit edges that cross each other. Additional clauses will rule that out.

394. Call these clauses *langford'''*(n). [Steven Prestwich described a similar method in *Trends in Constraint Programming* (Wiley, 2007), 269–274.] Typical results are:

	variables	clauses	Algorithm D	Algorithm L	Algorithm C	
langford'''(9)	206	1157	131 Mμ	18 Mμ	22 Mμ	(UNSAT)
langford'''(13)	403	2935	1425 Gμ	44 Gμ	483 Gμ	(UNSAT)
langford'''(16)	584	4859	713 Kμ	42 Mμ	343 Kμ	(SAT)
langford'''(64)	7352	120035	(huge)	(big)	71 Mμ	(SAT)

395. The color of each vertex v gets binary axiom clauses $(\bar v^{j+1} \vee v^j)$ for $1 \le j < d-1$, as in (164). And for each edge $u \text{ — } v$ in the graph, we want d clauses $(\bar u^{j-1} \vee u^j \vee \bar v^{j-1} \vee v^j)$ for $1 \le j \le d$, omitting $\bar u^0$ and $\bar v^0$ when $j=1$, u^d and v^d when $j=d$.

[The surprising usefulness of order encoding in graph coloring was first noticed by N. Tamura, A. Taga, S. Kitagawa, and M. Banbara in *Constraints* **14** (2009), 254–272.]

396. First we have $(\bar x^{j+1} \vee x^j)$ and $(\overline{\hat x^{j+1}} \vee \hat x^j)$ for $1 \le j < d$. Then we have "channeling" clauses to ensure that $j \le x < j+1 \iff j\pi \le x\pi < j\pi + 1$ for $0 \le j < d$:

$$(\bar x^j \vee x^{j+1} \vee \hat x^{j\pi}) \wedge (\bar x^j \vee x^{j+1} \vee \overline{\hat x^{j\pi+1}}) \wedge (\overline{\hat x^{j\pi}} \vee \hat x^{j\pi+1} \vee x^j) \wedge (\overline{\hat x^{j\pi}} \vee \hat x^{j\pi+1} \vee \bar x^{j+1}).$$

(These clauses should be either shortened or omitted in boundary cases, because x^0 and $\hat x^0$ are always true, while x^d and $\hat x^d$ are always false. We obtain $6d-8$ clauses for each x.)

With such clauses for every vertex of a graph, together with clauses based on adjacent vertices and cliques, we obtain encodings for n-coloring the $n \times n$ queen graph that involve $2(n^3-n^2)$ variables and $\frac{5}{3}n^4 + 4n^3 + O(n^2)$ clauses, compared to n^3-n^2 variables and $\frac{5}{3}n^4 - n^3 + O(n^2)$ clauses with single cliques and (162) alone. Typical running times with Algorithm C and single cliques are 323 Kμ, 13.1 Mμ, 706 Gμ for $n = 7, 8, 9$; with double clique-ing they become 252 Kμ, 1.97 Mμ, 39.8 Gμ, respectively.

The double clique hints turn out to be mysteriously ineffective when π is the standard organ-pipe permutation $(0\pi, 1\pi, \ldots, (d-1)\pi) = (0, 2, 4, \ldots, 5, 3, 1)$ instead of its inverse. Random choices of π when $n = 8$ yielded significant improvement almost half the time, in the author's experiments; but they had negligible effect in 1/3 of the cases.

Notice that the example π for $d = 4$ yields $x^1 = \bar x_0$, $x^3 = x_3$, $\hat x^1 = \bar x_2$, $\hat x_3 = x_1$. Hence the direct encoding is essentially present as part of this redundant representation, and the hints $(\bar u^3 \vee \bar v^3) \wedge (u^1 \vee v^1) \vee (\overline{\hat u^3} \vee \overline{\hat v^3}) \wedge (\hat u^1 \vee \hat v^1)$ for 2-cliques $\{u, v\}$ are equivalent to (16). But the hints $(u^2 \vee v^2 \vee w^2) \wedge (\bar u^2 \vee \bar v^2 \vee \bar w^2) \wedge (\hat u^2 \vee \hat v^2 \vee \hat w^2) \wedge (\overline{\hat u^2} \vee \overline{\hat v^2} \vee \overline{\hat w^2})$ that apply when $\{u, v, w\}$ is a triangle give additional logical power.

Steiner tree packing
Grötschel
Martin
Weismantel
langford'''(n)
Prestwich
axiom clauses
Tamura
Taga
Kitagawa
Banbara
channeling
author
direct encoding
triangle

397. There are $(p-2)d$ binary clauses $(\bar{y}_j^{i+1} \vee y_j^i)$ for $1 \le i < p-1$, together with the $(2p-2)d$ clauses $(\bar{x}_i^j \vee x_i^{j+1} \vee y_j^i) \wedge (\bar{x}_{i-1}^j \vee x_{i-1}^{j+1} \vee \bar{y}_j^i)$ for $1 \le i < p$, all for $0 \le j < d$. The hint clauses $(x_0^{p-1} \vee \cdots \vee x_{p-1}^{p-1}) \wedge (\bar{x}_0^{d-p+1} \vee \cdots \vee \bar{x}_{p-1}^{d-p+1})$ are also valid.

(This setup corresponds to putting p pigeons into d holes, so we can usually assume that $p \le d$. If $p \le 4$ it is better to use $\binom{p}{2}d$ clauses as in exercise 395. Notice that we obtain an interesting representation of *permutations* when $p = d$. In that case y is the inverse permutation; hence $(2d-2)p$ additional clauses corresponding to $y_j = i \implies x_i = j$ are also valid, as well as two hint clauses for y.)

A related idea, but with direct encoding of the x's, was presented by I. Gent and P. Nightingale in *Proceedings of the International Workshop on Modelling and Reformulating Constraint Satisfaction Problems* **3** (2004), 95–110.

398. We could construct $(3p-4)d$ binary clauses that involve y_j^i, as in exercise 397. But it's better just to have $(3p-6)d$ clauses for the at-most-one constraints $x_{0k} + x_{1k} + \cdots + x_{(p-1)k} \le 1$, $0 \le k < d$.

399. (a) $d^2 - t$ preclusion clauses (binary); or $2d$ support clauses (total length $2(d+t)$).

(b) If unit propagation derives $\bar{v}_j$ from $(\bar{u}_i \vee \bar{v}_j)$, we knew u_i; hence (17) gives $\bar{u}_{i'}$ for all $i' \ne i$, and $\bar{v}_j$ follows from the support clause that contains it.

(c) If unit propagation derives $\bar{v}_j$ from its support clause, we knew $\bar{u}_i$ for all $i \ne j$; hence (15) gives u_j, and $\bar{v}_j$ follows from (16). Or if unit propagation derives u_i from that support clause, we knew v_j and $\bar{u}_{i'}$ for all $i' \notin \{i, j\}$; hence $\bar{u}_j$ from (16), u_i from (15).

(d) A trivial example has no legal pairs; then unit propagation never gets started from binary preclusions, but the (unit) support clauses deduce all. A more realistic example has $d = 3$ and all pairs legal except $(1,1)$ and $(1,2)$, say; then we have $(15) \wedge (17) \wedge (\bar{u}_1 \vee \bar{v}_1) \wedge (\bar{u}_1 \vee \bar{v}_2) \wedge (\bar{v}_3) \nvdash_1 \bar{u}_1$ but $(15) \wedge (17) \wedge (\bar{u}_1 \vee v_3) \wedge (\bar{v}_3) \vdash_1 \bar{u}_1$.

[Preclusion was introduced by S. W. Golomb and L. D. Baumert, *JACM* **12** (1965), 521–523. The support encoding was introduced by I. P. Gent, *European Conf. on Artificial Intelligence* **15** (2002), 121–125, based on work of S. Kasif, *Artificial Intelligence* **45** (1990), 275–286.]

400. This problem has n variables $q_1, \ldots, q_n$ with n values each; thus there are n^2 Boolean values, with $q_{ij} = [q_i = j] =$ [there's a queen in row i and column j]. The constraint between q_i and q_j is that $q_i \notin \{q_j, q_j + i - j, q_j - i + j\}$; so it turns out that there are n at-least-one clauses, plus $(n^3 - n^2)/2$ at-most-one clauses, plus either $n^3 - n^2$ support clauses or $n^3 - n^2 + \binom{n}{3}$ preclusion clauses. In this problem each support clause has at least $n - 2$ literals, so the support encoding is much larger.

Since the problem is easily satisfiable, it makes sense to try WalkSAT. When $n = 20$, Algorithm W typically finds a solution from the preclusion clauses after making fewer than 500 flips; its running time is about 500 Kμ, including about 200 Kμ just to read the input. With the support clauses, however, it needs about 10 times as many flips and consumes about 20 times as many mems, before succeeding.

Algorithm L is significantly worse: It consumes 50 Mμ with preclusion clauses, 11 Gμ with support clauses. Algorithm C is the winner, with about 400 Kμ (preclusion) versus 600 Kμ (support).

Of course $n = 20$ is pretty tame; let's consider $n = 100$ queens, when there are 10,000 variables and more than a million clauses. Algorithm L is out of the picture; in the author's experiments, it showed no indication of being even close to a solution after 20 Tμ! But Algorithm W solves that problem in 50 Mμ, via preclusion, after making only about 5000 flips. Algorithm C wins again, polishing it off in 29 Mμ. With

pigeons
permutations
inverse permutation
direct encoding
Gent
Nightingale
at-most-one constraints
Golomb
Baumert
Gent
Kasif
WalkSAT
author

the support clauses, nearly 100 million literals need to be input, and Algorithm W is hopelessly inefficient; but Algorithm C is able to finish after about 200 Mμ.

The preclusion clauses actually allow us to omit the at-most-one clauses in this problem, because two queens in the same row will be ruled out anyway. This trick improves the run time when $n = 100$ to 35 Mμ for Algorithm W.

We can also append support clauses for the columns as well as the rows. This idea roughly halves the search space, but it gives no improvement because twice as many clauses must be handled. *Bottom line:* Support clauses don't support n queens well.

(However, if we seek *all* solutions to the n queens problem instead of stopping with the first one, using a straightforward extension of Algorithm D (see exercise 122), the support clauses proved to be definitely better in the author's experiments.)

401. (a) $y^j = x^{2j-1}$. (b) $z^j = x^{3j-1}$. In general $w = \lfloor (x+a)/b \rfloor \iff w^j = x^{bj-a}$.

402. (a) $\bigwedge_{j=1}^{\lfloor d/2 \rfloor} (\bar{x}^{2j-1} \vee x^{2j})$; (b) $\bigwedge_{j=1}^{\lceil d/2 \rceil} (\bar{x}^{2j-2} \vee x^{2j-1})$; omit $\bar{x}^0$ and x^d.

403. (a) $\bigwedge_{j=1}^{d-1} (\bar{x}^j \vee \bar{y}^j \vee z^j)$; (b) $\bigwedge_{j=1}^{d-1} ((\bar{x}^j \vee z^j) \wedge (\bar{y}^j \vee z^j))$; (c) $\bigwedge_{j=1}^{d-1} ((x^j \vee \bar{z}^j) \wedge (y^j \vee \bar{z}^j))$; (d) $\bigwedge_{j=1}^{d-1} (x^j \vee y^j \vee \bar{z}^j)$.

404. (a) $\bigwedge_{j=0}^{d-1} (\bar{x}^j \vee x^{j+1} \vee \bar{y}^{j+1-a} \vee y^{j+a})$. (As usual, omit literals with superscripts ≤ 0 or $\geq d$. If $a > 1$ this encoding is unsymmetrical, with one clause for each value of x.)

(b) $\bigwedge_{j=0}^{d-a} ((p \vee \bar{x}^j \vee y^{j+a}) \wedge (\bar{p} \vee x^{j+a} \vee \bar{y}^j))$; p is the auxiliary variable.

405. (a) If $a < 0$ we can replace ax by $(-a)\bar{x}$ and c by $c + a - ad$, where $\bar{x}$ is given by (165). A similar reduction applies if $b < 0$. Cases with a, b, or $c = 0$ are trivial.

(b) We have $13x + 8\bar{y} \leq 63 \iff$ not $13x + 8\bar{y} \geq 64 \iff$ not (P_0 or ... or P_{d-1}) $\iff$ not P_0 and ... and not P_{d-1}, where P_j = '$x \geq j$ and $\bar{y} \geq \lceil (64-13j)/8 \rceil$'. This approach yields $\bigwedge_{j=0}^{7} (\bar{x}^j \vee y^{8-\lceil (64-13j)/8 \rceil})$, which simplifies to $(\bar{x}^1 \vee y^1) \wedge (\bar{x}^2 \vee y^3) \wedge (\bar{x}^3 \vee y^4) \wedge (\bar{x}^4 \vee y^6) \wedge (\bar{x}^5)$. (Notice that we could have defined P_j = '$\bar{y} \geq j$ and $x \geq \lceil (64-8j)/13 \rceil$' instead, thereby obtaining the less efficient encoding $(\bar{x}^5) \wedge (y^7 \vee \bar{x}^5) \wedge (y^6 \vee \bar{x}^4) \wedge (y^5 \vee \bar{x}^4) \wedge (y^4 \vee \bar{x}^3) \wedge (y^3 \vee \bar{x}^2) \wedge (y^2 \vee \bar{x}^2) \wedge (y^1 \vee \bar{x}^1)$; it's better to discriminate on the variable with the larger coefficient.)

(c) Similarly, $13\bar{x} + 8y \leq 90$ gives $(x^5 \vee \bar{y}^7) \wedge (x^4 \vee \bar{y}^5) \wedge (x^3 \vee \bar{y}^4) \wedge (x^2 \vee \bar{y}^2) \wedge (x^1)$. (The (x, y) pairs legal for both (b) and (c) are $(1,1)$, $(2,3)$, $(3,4)$, $(4,6)$.)

(d) $\bigwedge_{j=\max(0,\lceil (c+1-b(d-1))/a \rceil)}^{\min(d-1,\lceil (c+1)/a \rceil)} (\bar{x}^j \vee \bar{y}^{\lceil (c+1-aj)/b \rceil})$, when $a \geq b > 0$ and $c \geq 0$.

406. (a) $(\bigwedge_{j=\lceil (a+1)/(d-1) \rceil}^{\lfloor \sqrt{a+1} \rfloor} (\bar{x}^j \vee \bar{y}^{\lceil (a+1)/j \rceil})) \wedge (\bigwedge_{j=\lceil (a+1)/(d-1) \rceil}^{\lceil \sqrt{a+1} \rceil - 1} (\bar{x}^{\lceil (a+1)/j \rceil} \vee \bar{y}^j))$.

(b) $(\bigwedge_{j=l+1}^{\lfloor \sqrt{a-1} \rfloor + 1} (x^j \vee y^{\lfloor (a-1)/(j-1) \rfloor + 1})) \wedge (\bigwedge_{j=l+1}^{\lceil \sqrt{a-1} \rceil} (x^{\lfloor (a-1)/(j-1) \rfloor + 1} \vee y^j)) \wedge (x^l) \wedge (y^l)$, where $l = \lfloor (a-1)/(d-1) \rfloor + 1$. [Both formulas belong to 2SAT.]

407. (a) We always have $\lfloor x/2 \rfloor + \lceil x/2 \rceil = x$, $\lfloor x/2 \rfloor + \lfloor y/2 \rfloor \leq \frac{x+y}{2} \leq \lfloor x/2 \rfloor + \lfloor y/2 \rfloor + 1$, and $\lceil x/2 \rceil + \lceil y/2 \rceil - 1 \leq \frac{x+y}{2} \leq \lceil x/2 \rceil + \lceil y/2 \rceil$. (Similar reasoning proves the correctness of Batcher's odd-even merge network; see Eq. 5.3.4–(3).)

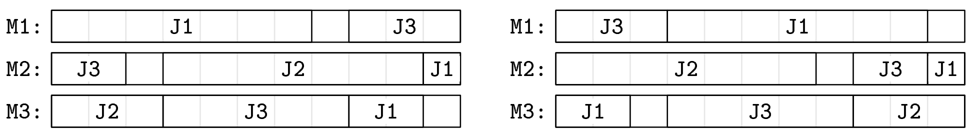
(b) Axiom clauses like (164) needn't be introduced for u and v, or even for z; so they aren't counted here, although they could be added if desired. Let $a_d = d^2 - 1$ be the number of clauses in the original method; then the new method has fewer clauses when $a_{\lceil d/2 \rceil} + a_{\lfloor d/2 \rfloor + 1} + 3(d-2) < a_d$, namely when $d \geq 7$. (The new method for $d = 7$ involves 45 clauses, not 48; but it introduces 10 new auxiliary variables.) Asymptotically, we can handle $d = 2^t + 1$ with $3t2^t + O(2^t) = 3d \lg d + O(d)$ clauses and $d \lg d + O(d)$ auxiliary variables.

(c) $x + y \geq z \iff (d-1-x) + (d-1-y) \leq (2d-2-z)$; so we can use the same method, but complemented (namely with $x^j \mapsto \bar{x}^{d-j}$, $y^j \mapsto \bar{y}^{d-j}$, $z^j \mapsto \bar{z}^{2d-1-j}$).

at-most-one
author
all solutions
2SAT
midpoint
Batcher
odd-even
merge network
Axiom clauses
complemented

[See N. Tamura, A. Taga, S. Kitagawa, and M. Banbara, *Constraints* **14** (2009), 254–272; R. Asín, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell, *Constraints* **16** (2011), 195–221.]

408. (a) No; makespan 11 is best, achievable as follows (or via left-right reflection):



(b) If j is the last job processed by machine i, that machine must finish at time $\le \sum_{k=1}^{n} w_{ik} + \sum_{k=1}^{m} w_{kj} - w_{ij}$, because j uses some other machine whenever i is idle. [See D. B. Shmoys, C. Stein, and J. Wein, *SICOMP* **23** (1994), 631.]

(c) Clearly $0 \le s_{ij} \le t - w_{ij}$. And if $ij \ne i'j'$ but $i = i'$ or $j = j'$, we must have either $s_{ij} + w_{ij} \le s_{i'j'}$ or $s_{i'j'} + w_{i'j'} \le s_{ij}$ whenever $w_{ij}w_{i'j'} \ne 0$.

(d) When $w_{ij} > 0$, introduce Boolean variables s_{ij}^k for $1 \le k \le t - w_{ij}$, with the axiom clauses $(\bar{s}_{ij}^{k+1} \vee s_{ij}^k)$ for $1 \le k < t - w_{ij}$. Then include the following clauses for all relevant i, j, i', and j' as in (c): For $0 \le k \le t + 1 - w_{ij} - w_{i'j'}$, assert $(\bar{p}_{iji'j'} \vee \bar{s}_{ij}^k \vee s_{i'j'}^{k+w_{ij}})$ if $ij < i'j'$ or $(p_{i'j'ij} \vee \bar{s}_{ij}^k \vee s_{i'j'}^{k+w_{ij}})$ if $ij > i'j'$, omitting $\bar{s}_{ij}^0$ in the first of these ternary clauses and omitting $s_{i'j'}^{t+1-w_{i'j'}}$ in the last.

[This method, introduced by N. Tamura, A. Taga, S. Kitagawa, and M. Banbara in *Constraints* **14** (2009), 254–272, was able to solve several open shop scheduling problems in 2008 that had resisted attacks by all other approaches.]

Since the left-right reflection of any valid schedule is also valid, we can also save a factor of two by arbitrarily choosing one of the p variables and asserting $(p_{iji'j'})$.

(e) Any schedule for W and T yields a schedule for $\lfloor W/k \rfloor$ and $\lceil T/k \rceil$, if we examine time slots 0, k, $2k$, $\ldots$. [With this observation we can narrow down the search for an optimum makespan by first working with simpler problems; the number of variables and clauses for $\lfloor W/k \rfloor$ and T/k is about $1/k$ times the number for W and T, and the running time also tends to obey this ratio. For example, the author solved a nontrivial 8×8 problem by first working with $\lfloor W/8 \rfloor$ and getting the respective results $(\mathrm{U}, \mathrm{S}, \mathrm{U})$ for $t = (128, 130, 129)$, where 'U' means "unsatisfiable" and 'S' means "satisfiable"; running times were about $(75, 10, 1250)$ megamems. Then with $\lfloor W/4 \rfloor$ it was $(\mathrm{S}, \mathrm{U}, \mathrm{U})$ with $t = (262, 260, 261)$ and runtimes $(425, 275, 325)$; with $\lfloor W/2 \rfloor$ it was $(\mathrm{U}, \mathrm{S}, \mathrm{U})$ with $t = (526, 528, 527)$ and runtimes $(975, 200, 900)$. Finally with the full W it was $(\mathrm{U}, \mathrm{S}, \mathrm{S})$ with $t = (1058, 1060, 1059)$ and runtimes $(2050, 775, 300)$, establishing 1059 as the optimum makespan while doing most of the work on small subproblems.]

Notes: Further savings are possible by noting that any clauses learned while proving that t is satisfiable are valid also when t is decreased. Difficult random problems can be generated by using the following method suggested by C. Guéret and C. Prins in *Annals of Operations Research* **92** (1999), 165–183: Start with work times w_{ij} that are as near equal as possible, having constant row and column sums s. Then choose random rows $i \ne i'$ and random columns $j \ne j'$, and transfer δ units of weight by setting $w_{ij} \leftarrow w_{ij} - \delta$, $w_{i'j} \leftarrow w_{i'j} + \delta$, $w_{ij'} \leftarrow w_{ij'} + \delta$, $w_{i'j'} \leftarrow w_{i'j'} - \delta$, where $\delta \ge w_{ij}$ and $\delta \ge w_{i'j'}$; this operation clearly preserves the row and column sums. Choose δ at random between $p \cdot \min\{w_{ij}, w_{i'j'}\}$ and $\min\{w_{ij}, w_{i'j'}\}$, where p is a parameter. The final weights are obtained after making r such transfers. Guéret and Prins suggested choosing $r = n^3$, and $p = .95$ for $n \ge 6$; but other choices give useful benchmarks too.

Tamura
Taga
Kitagawa
Banbara
Asín
Nieuwenhuis
Oliveras
Rodríguez-Carbonell
Shmoys
Stein
Wein
Tamura
Taga
Kitagawa
Banbara
symmetry breaking
Knuth
Guéret
Prins

409. (a) If $S \subseteq \{1,\ldots,r\}$, let $\Sigma_S = \sum_{j\in S} a_j$. We can assume that job n runs on machines 1, 2, 3 in that order. So the minimum makespan is $2w_{2n}+x$, where x is the smallest Σ_S that is $\geq \lceil (a_1+\cdots+a_r)/2\rceil$. (The problem of finding such an S is well known to be NP-hard [R. M. Karp, *Complexity of Computer Computations* (New York: Plenum, 1972), 97–100]; hence the open shop scheduling problem is NP-complete.)

(b) Makespan $w_{2n}+w_{4n}$ is achievable if and only if $\Sigma_S = (a_1+\cdots+a_r)/2$ for some S. Otherwise we can achieve makespan $w_{2n}+w_{4n}+1$ by running jobs $1, \ldots, n$ in order on machine 1 and letting $s_{3(n-1)} = 0$, $s_{4n} = w_{2n}$; also $s_{2j} = w_{2n}+w_{4n}$, if machine 1 is running job j at time w_{2n}. The other jobs are easily scheduled.

(c) $\lfloor 3n/2\rfloor - 2$ time slots are clearly necessary and sufficient. (If all row and column sums of W are equal to s, can the minimum makespan be $\geq \frac{3}{2}s$?)

(d) The "tight" makespan s is always achievable: By renumbering the jobs we can assume that $a_j \leq b_j$ for $1\leq j\leq k$, $a_j \geq b_j$ for $k<j\leq n$, $b_1 = \max\{b_1,\ldots,b_k\}$, $a_n = \max\{a_{k+1},\ldots,a_n\}$. Then if $b_n \geq a_1$, machine 1 can run jobs $(1,\ldots,n)$ in order while machine 2 runs $(n,1,\ldots,n-1)$; otherwise $(2,\ldots,n,1)$ and $(1,\ldots n)$ suffice.

If $a_1+\cdots+a_n \neq b_1+\cdots+b_n$, we can increase a_n or b_n to make them equal. Then we can add a "dummy" job with $a_{n+1} = b_{n+1} = \max\{a_1+b_1,\ldots,a_n+b_n\} \dot{-} s$, and obtain an optimum schedule in $O(n)$ steps as explained above.

Results (a), (b), (d) are due to T. Gonzalez and S. Sahni, who introduced and named the open shop scheduling problem in *JACM* **23** (1976), 665–679. Part (c) is a subsequent observation and open problem due to Gonzalez (unpublished).

410. Using half adders and full adders as we did in (23) allows us to introduce intermediate variables w_j such that $(x_2x_1x_0)_2 + (x_2x_1x_000)_2 + (x_2x_1x_0000)_2 + (\bar y_2\bar y_1\bar y_0000)_2 \leq (w_7w_6\ldots w_0)_2$, and then to require $(\bar w_7)\wedge(\bar w_6)$. In slow motion, we successively compute $(c_0z_0)_2 \geq x_0+x_1$, $(c_1z_1)_2 \geq x_0+x_1+\bar y_0$, $(c_2z_2)_2 \geq c_0+z_1$, $(c_3z_3)_2 \geq x_1+x_2+\bar y_1$, $(c_4z_4)_2 \geq c_1+c_2+z_3$, $(c_5z_5)_2 \geq x_2+\bar y_2+c_3$, $(c_6z_6)_2 \geq c_4+z_5$, $(c_7z_7)_2 \geq c_5+c_6$; then $w_7w_6\ldots w_0 = c_7z_7z_6z_4z_2z_0x_1x_0$. In slower motion, each step $(c_iz_i)_2 \geq u+v$ expands to $z_i \geq u\oplus v$, $c_i \geq u\wedge v$; each step $(c_iz_i)_2 \geq t+u+v$ expands to $s_i \geq t\oplus u$, $p_i \geq t\wedge u$, $z_i \geq v\oplus s$, $q_i \geq v\wedge s$, $c_i \geq p_i\vee q_i$. And at the clause level, $t\geq u\wedge v \iff (t\vee\bar u\vee\bar v)$; $t\geq u\vee v \iff (t\vee\bar u)\wedge(t\vee\bar v)$; $t\geq u\oplus v \iff (t\vee\bar u\vee v)\wedge(t\vee u\vee\bar v)$. [Only about half of (24) is needed when inequalities replace equalities. Exercise 42 offers improvements.]

We end up with 44 binary and ternary clauses; 10 of them can be omitted, because z_0, z_2, z_4, z_6, and z_7 are pure literals, and the clause for c_7 can be omitted if we simply require $c_5 = c_6 = 0$. But the order encoding of exercise 405 is clearly much better. The log encoding becomes attractive only with larger integers, as in the following exercise. [See J. P. Warners, *Information Processing Letters* **68** (1998), 63–69.]

411. Use $m+n$ new variables to represent an auxiliary number $w = (w_{m+n}\ldots w_1)_2$. Form clauses as in exercise 41 for the product $xy = w$; but retain only about half of the clauses, as in answer 410. The resulting $9mn-5m-10n$ clauses are satisfiable if $w = xy$; and we have $w\geq xy$ whenever they are satisfiable. Now add $3m+3n-2$ further clauses as in (169) to ensure that $z\geq w$. The case $z\leq xy$ is similar.

412. Mixed-radix representations are also of interest in this connection. See, for example, N. Eén and N. Sörensson, *J. Satisfiability, Bool. Modeling and Comp.* **2** (2006), 1–26; T. Tanjo, N. Tamura, and M. Banbara, *LNCS* **7317** (2012), 456–462.

413. Eliminating first a_{n-1}, then a_{n-2}, etc., yields 2^n-1 clauses. (The analogous result for $x_1\ldots x_n < y_1\ldots y_n$ is $2^n+2^{n-1}+1$. A preprocessor will probably eliminate a_{n-1}.)

subset sum problem
Karp
NP-complete
Gonzalez
Sahni
half adders
full adders
Tseytin encoding, half
pure literals
order encoding
Warners
Mixed-radix
Eén
Sörensson
Tanjo
Tamura
Banbara
preprocessor

(pure) literal
Biere
Brummayer
blocked
subsumed clauses
blocked clause
pure literal
failed literal tests

414. Construct clauses for $1 \le k \le n$ that represent 'a_{k-1} implies $x_k < y_k + a_k$':

$$\Bigl(\bar a_{k-1} \vee \bigvee_{j=1}^{d-1} (\bar x_k^j \vee y_k^j)\Bigr) \wedge \Bigl(\bar a_{k-1} \vee a_k \vee \bigvee_{j=0}^{d-1} (\bar x_k^j \vee y_k^{j+1})\Bigr), \quad \text{omitting } \bar x_k^0 \text{ and } y_k^d;$$

also omit $\bar a_0$. For the relation $x_1 \ldots x_n \le y_1 \ldots y_n$ we can omit the d clauses that contain the (pure) literal a_n. But for $x_1 \ldots x_n < y_1 \ldots y_n$, we want $a_n = 0$; so we omit a_n and the $d-1$ clauses $(\bar a_{n-1} \vee \bar x_n^j \vee y_n^j)$.

415. There's only one, namely $\bigwedge_{\sigma_1,\ldots,\sigma_n \in \{-1,1\}} (\sigma_1 x_1 \vee \sigma_1 y_1 \vee \cdots \vee \sigma_n x_n \vee \sigma_n y_n)$. *Proof:* Some clause must contain only positive literals, because $f(0,\ldots,0) = 0$. This clause must be $(x_1 \vee y_1 \vee \cdots \vee x_n \vee y_n)$; otherwise it would be false in cases where f is true. A similar argument shows that *every* clause $(\sigma_1 x_1 \vee \sigma_1 y_1 \vee \cdots \vee \sigma_n x_n \vee \sigma_n y_n)$ must be present. And no clause for f can contain both x_j and $\bar y_j$, or both $\bar x_j$ and y_j.

416. The other clauses are $\bigwedge_{i=1}^{m} \bigl((u_i \vee \bar v_i \vee \bar a_0) \wedge (\bar u_i \vee v_i \vee \bar a_0)\bigr)$ and $(a_0 \vee a_1 \vee \cdots \vee a_n)$. [See A. Biere and R. Brummayer, *Proceedings, International Conference on Formal Methods in Computer Aided Design* **8** (IEEE, 2008), 4 pages [FMCAD 08].]

417. The four clauses $(\bar s \vee \bar t \vee u) \wedge (\bar s \vee t \vee v) \wedge (s \vee \bar t \vee \bar u) \wedge (s \vee t \vee \bar v)$ ensure that s is true if and only if $t ?\, u \colon v$ is true. But we need only the first two of these, as in (174), when translating a branching program, because the other two are blocked in the initial step. Removing them makes the other two blocked on the second step, etc.

418. A suitable branching program for h_n when $n = 3$, beginning at I_{11}, is $I_{11} = (\bar 1 ?\, 21 \colon 22)$, $I_{21} = (\bar 2 ?\, 31 \colon 32)$, $I_{22} = (\bar 2 ?\, 32 \colon 33)$, $I_{31} = (\bar 3 ?\, 0 \colon 42)$, $I_{32} = (\bar 3 ?\, 42 \colon 43)$, $I_{33} = (\bar 3 ?\, 43 \colon 1)$, $I_{42} = (\bar 1 ?\, 0 \colon 1)$, $I_{43} = (\bar 2 ?\, 0 \colon 1)$. It leads via (174) to the following clauses for row i, $1 \le i \le m$: $(r_{i,1,1})$; $(\bar r_{i,k,j} \vee x_{ik} \vee r_{i,k+1,j}) \wedge (\bar r_{i,k,j} \vee \bar x_{ik} \vee r_{i,k+1,j+1})$, for $1 \le j \le k \le n$; $(\bar r_{i,n+1,1}) \wedge (r_{i,n+1,n+1})$ and $(\bar r_{i,n+1,j+1} \vee x_{ij})$ for $1 \le j < n$. Also the following clauses for column j, $1 \le j \le n$: $(c_{i,1,1})$; $(\bar c_{j,k,i} \vee x_{kj} \vee c_{j,k+1,i}) \wedge (\bar c_{j,k,i} \vee \bar x_{kj} \vee c_{j,k+1,i+1})$, for $1 \le i \le k \le m$; $(\bar c_{j,m+1,1}) \wedge (c_{j,m+1,m+1})$ and $(\bar c_{j,m+1,i+1} \vee x_{ij})$ for $1 \le i < m$.

419. (a) There are exactly $n-2$ solutions: $x_{ij} = [j=1][i \ne m-1] + [j=2][i=m-1] + [j=k][i=m-1]$, for $2 < k \le n$.

(b) There are exactly $m-2$ solutions: $\bar x_{ij} = [j>1][i=m-1] + [j=1][i=m-2] + [j=1][i=k]$, for $1 \le k < m-2$ or $k = m$.

420. Start via (24) with $(\bar x_1 \vee x_2 \vee s) \wedge (x_1 \vee \bar x_2 \vee s) \wedge (x_1 \vee x_2 \vee \bar s) \wedge (\bar x_1 \vee \bar x_2 \vee \bar s)$; $(x_1 \vee \bar c) \wedge (x_2 \vee \bar c) \wedge (\bar x_1 \vee \bar x_2 \vee c)$; $(\bar s \vee x_3 \vee t) \wedge (s \vee \bar x_3 \vee t) \wedge (s \vee x_3 \vee \bar t) \wedge (\bar s \vee \bar x_3 \vee \bar t)$; $(s \vee \bar c') \wedge (x_3 \vee \bar c') \wedge (\bar s \vee \bar x_3 \vee c')$; $(\bar c) \wedge (\bar c')$. Propagate $(\bar c)$ and $(\bar c')$, obtaining $(\bar x_1 \vee \bar x_2) \wedge (\bar s \vee \bar x_3)$; remove subsumed clauses $(\bar x_1 \vee \bar x_2 \vee \bar s)$, $(\bar s \vee \bar x_3 \vee \bar t)$; remove blocked clause $(s \vee x_3 \vee \bar t)$; remove clauses containing the pure literal t; rename s to a_1.

421. Start via (173) with $(\bar a_5 \vee x_1 \vee a_4) \wedge (\bar a_5 \vee \bar x_1 \vee a_3) \wedge (\bar a_4 \vee \bar x_2 \vee a_2) \wedge (\bar a_3 \vee x_2 \vee a_2) \wedge (\bar a_3 \vee \bar x_2) \wedge (\bar a_2 \vee \bar x_3) \wedge (a_5)$. Propagate (a_5).

422. (a) x_1 implies $\bar x_2$, then a_1, then $\bar x_3$; x_2 implies $\bar x_1$, then a_1, then $\bar x_3$.

(b) x_1 implies a_3, then $\bar x_2$, then a_2, then $\bar x_3$; x_2 implies $\bar a_3$, then $\bar x_1$, a_4, a_2, $\bar x_3$.

423. No; consider $x_1 ?\, (x_2 ?\, x_3 \colon x_4) \colon (x_2 ?\, x_4 \colon x_3)$ with $L = (\bar x_3) \wedge (\bar x_4)$. [But a *forcing* encoding *can* always be constructed, via the extra clauses defined in exercise 436. Notice that, in the presence of failed literal tests, weak forcing corresponds to forcing.]

424. The clause $\bar 1\bar 3\bar 4$ is redundant (in the presence of $\bar 1 \bar 2 \bar 3$ and $2\bar 3\bar 4$); it cannot be omitted, because $\{\bar 2\bar 3, 2\bar 3, 12\} \nvdash_1 \bar 3$. The clause $2\bar 3\bar 4$ is also redundant (in the presence of $\bar 1\bar 3\bar 4$ and 12); it *can* be omitted, because $\{\bar 1\bar 4, 34, 1\} \vdash_1 \bar 4$, $\{\bar 1\bar 3, 34, 1\} \vdash_1 \bar 3$, and $\{\bar 1\bar 2, \bar 1, 12\} \vdash_1 2$.

425. If x is in the core, $F \vdash_1 x$, because Algorithm 7.1.1C does unit propagation. Otherwise F is satisfied when all core variables are true and all noncore variables are false.

426. (a) True. Suppose the clauses involving a_m are $(a_m \vee \alpha_i)$ for $1 \le i \le p$ and $(\bar{a}_m \vee \beta_j)$ for $1 \le j \le q$; then G contains the pq clauses $(\alpha_i \vee \beta_j)$ instead. If $F \,|\, L \vdash_1 l$ we want to prove that $G \,|\, L \vdash_1 l$. This is clear if unit propagation from $F \,|\, L$ doesn't involve a_m. Otherwise, if $F \,|\, L \vdash_1 a_m$, unit propagation has falsified some α_i; every subsequent propagation step from $F \,|\, L$ that uses $(\bar{a}_m \vee \beta_j)$ can use $(\alpha_i \vee \beta_j)$ in a propagation step from $G \,|\, L$. A similar argument applies when $F \,|\, L \vdash_1 \bar{a}_m$.

(Incidentally, variable elimination also preserves "honesty.")

(b) False. Let $F = (x_1 \vee x_2 \vee a_1) \wedge (x_1 \vee x_2 \vee \bar{a}_1)$, $L = \bar{x}_1$ or $\bar{x}_2$.

427. Suppose $n = 3m$, and let f be the symmetric function $[\nu x{<}m \text{ or } \nu x{>}2m]$. The prime clauses of f are the $N = \binom{n}{m,m,m} \sim 3^{n+3/2}/(2\pi n)$ ORs of m positive literals and m negative literals. There are $N' = \binom{n}{m-1,m,m+1} = \frac{m}{m+1}N$ ways to specify that $x_{i_1} = \cdots = x_{i_m} = 1$ and $x_{i_{m+1}} = \cdots = x_{i_{2m-1}} = 0$; and this partial assignment implies that $x_j = 1$ for $j \notin \{i_1, \ldots, i_{2m-1}\}$. Therefore at least one of the $m+1$ clauses $(\bar{x}_{i_1} \vee \cdots \vee \bar{x}_{i_m} \vee x_{i_{m+1}} \vee \cdots \vee x_{i_{2m-1}} \vee x_j)$ must be present in any set of prime clauses that forces f. By symmetry, any such set must include at least N'/m prime clauses.

On the other hand, f is characterized by $O(n^2)$ forcing clauses (see answer 436).

428. (a) $(y \vee z_{j1} \vee \cdots \vee z_{jd})$ for $1 \le j \le n$; $(\bar{x}_{ij} \vee \bar{z}_{ik} \vee \bar{z}_{jk})$ for $1 \le i < j \le n$, $1 \le k \le d$.

(b) Imagine a circuit with $2N(N+1)$ gates g_{lt}, one for each literal l of G_{nd} and for each $0 \le t \le N$, meaning that literal l is known to be true after t rounds of unit propagation, if we start with given values of the x_{ij} variables only. Thus we set $g_{l0} \leftarrow 1$ if $l = x_{ij}$ and x_{ij} is true, or if $l = \bar{x}_{ij}$ and x_{ij} is false; otherwise $g_{l0} \leftarrow 0$. And

$$g_{l(t+1)} \leftarrow g_{lt} \vee \bigvee \{ g_{\bar{l}_1 t} \wedge \cdots \wedge g_{\bar{l}_k t} \mid (l \vee l_1 \vee \cdots \vee l_k) \in G_{nd} \}, \quad \text{for } 1 \le t < N.$$

Given values of the x_{ij}, the literal y is implied if and only if the graph has no d-coloring; and at most N rounds make progress. Thus there's a monotone chain for $g_{yN} = \bar{f}_{nd}$.

[This exercise was suggested by S. Buss and R. Williams in 2014, based on a similar construction by M. Gwynne and O. Kullmann.]

429. Let Σ_k be the sum of the assigned x's in leaves descended from node k. Unit propagation will force $b_j^k \leftarrow 1$ for $1 \le j \le \Sigma_k$, moving from leaves toward the root. Then it will force $b_j^k \leftarrow 0$ for $j = \Sigma_k + 1$, moving downwards from the root, because $r = \Sigma_2 + \Sigma_3$ and because (21) starts this process when $k = 2$ or 3.

430. Imagine boundary conditions as in answer 26, and assume that $x_{j_1}, \ldots, x_{j_r}$ have been assigned 1, where $j_1 < \cdots < j_r$. Unit propagation forces $s^k_{j_k+1-k} \leftarrow 1$ for $1 \le k \le r$; then it forces $s^k_{j_k-k} \leftarrow 0$ for $r \ge k \ge 1$. So unassigned x's are forced to zero.

431. Equivalently $x_1 + \cdots + x_m + \bar{y}_1 + \cdots + \bar{y}_n \le n$; so we can use (18)–(19) or (20)–(21).

432. The clauses of answer 404(b) can be shown to be forcing. But not those of 404(a) when $a > 1$; for example, if $a = 2$ and we assume $\bar{x}^2$, unit propagation doesn't yield y^2.

433. Yes. Imagine, for example, the partial assignment $x = 1{*}{*}{*}10{*}{*}1$, $y = 10{*}00{*}1{*}{*}$. Then y_3 must be 1; otherwise we'd have $10010001 \le x \le y \le 100001111$. In this situation unit propagation from the clauses that correspond to $1 \le \langle a_1 0 1 \rangle$, $a_1 \le \langle a_2 \bar{x}_2 0 \rangle$, $a_2 \le \langle a_3 \bar{x}_3 y_3 \rangle$, $a_3 \le \langle a_4 \bar{x}_4 0 \rangle$, $a_4 \le \langle a_5 0 0 \rangle$ forces $a_1 = 1$, $a_2 = 1$, $a_4 = 0$, $a_3 = 0$, $y_3 = 1$.

In general if a given partial assignment is consistent with $x \le y$, we must have $x{\downarrow} \le y{\uparrow}$, where $x{\downarrow}$ and $y{\uparrow}$ are obtained from x and y by changing all unassigned variables to 0 and 1, respectively. If that partial assignment forces some y_j to a

unit propagation
honesty
symmetric Boolean functions
prime clauses
Buss
Williams
Gwynne
Kullmann

particular value, the value must be 1; and we must in fact have $x{\downarrow} > y'{\uparrow}$, where y' is like y but with $y_j = 0$ instead of $y_j = *$. If $x_j \neq 1$, unit propagation will force $a_1 = \cdots = a_{j-1} = 1$, $a_k = \cdots = a_j = 0$, $y_j = 1$, for some $k \geq j$.

Similar remarks apply when x_i is forced, because $x \leq y \iff \bar{y} \leq \bar{x}$.

n.f.: not falsified
Bacchus

434. (a) Clearly p_k is equivalent to $\bar{x}_1 \wedge \cdots \wedge \bar{x}_k$, q_k is equivalent to $\bar{x}_k \wedge \cdots \wedge \bar{x}_n$, and r_k implies that a run of exactly l 1s begins at x_k.

(b) When $l = 1$, if $x_k = 1$ unit propagation will imply $\bar{p}_j$ for $j \geq k$ and $\bar{q}_j$ for $j \leq k$, hence $\bar{r}_j$ for $j \neq k$; then r_k is forced, making $x_j = 0$ for all $j \neq k$. Conversely, $x_j = 0$ forces $\bar{r}_j$; if this holds for all $j \neq k$, then r_k is forced, making $x_k = 1$.

But when $l = 2$ and $n = 3$, the clauses fail to force $x_2 = 1$ by unit propagation. They also fail to force $x_1 = 0$ when we have $l = 2$, $n = 4$, and $x_3 = 1$.

435. The following construction with $O(nl)$ clauses is satisfactory when l is small: Begin with the clauses for p_k and q_k (but not r_k) in exercise 434(a); include also $(\bar{x}_k \vee p_{k-l})$ for $l < k \leq n$, and $(\bar{x}_k \vee q_{k+l})$ for $1 \leq k \leq n-l$. Append $(\bar{p}_{k-l} \vee \bar{q}_{k+l} \vee x_k)$ for $1 \leq k \leq n$, omitting $\bar{p}_j$ for $j < 1$ and omitting $\bar{q}_j$ for $j > n$. Finally, append

$$(x_k \vee \bar{x}_{k+1} \vee x_{k+d}) \qquad \text{for } 0 \leq k < n \text{ and } 1 < d < l, \tag{$*$}$$

omitting x_j when $j < 1$ or $j > n$.

To reduce to $O(n \log l)$ clauses, suppose $2^{e+1} < l \leq 2^{e+2}$, where $e \geq 0$. The clauses $(*)$ can be replaced by $(\bar{x}_k \vee \bar{y}_k^{(e)} \vee \bar{z}_k^{(e)})$ for $1 \leq k \leq n$, if $\bar{x}_{k-d}$ implies $y_k^{(e)}$ for $1 \leq d \leq \lfloor l/2 \rfloor$ and $\bar{x}_{k+d}$ implies $z_k^{(e)}$ for $1 \leq d \leq \lceil l/2 \rceil$. And to achieve the latter, we introduce clauses $(\bar{y}_k^{(t)} \vee y_k^{(t+1)})$, $(\bar{y}_{k-2^t}^{(t)} \vee y_k^{(t+1)})$, $(\bar{z}_k^{(t)} \vee z_k^{(t+1)})$, $(\bar{z}_{k+2^t}^{(t)} \vee z_k^{(t+1)})$, $(x_{k-1} \vee y_k^{(0)})$, $(x_{k+2^e-1-\lfloor l/2 \rfloor} \vee y_k^{(0)})$, $(x_{k+1} \vee z_k^{(0)})$, $(x_{k-2^e+1+\lceil l/2 \rceil} \vee z_k^{(0)})$, for $1 \leq k \leq n$ and $0 \leq t < e$, always omitting x_j or $\bar{y}_j$ or $\bar{z}_j$ when $j < 1$ or $j > n$.

436. Let the variables q_k for $0 \leq k \leq n$ and $q \in Q$ represent the sequence of states, and let t_{kaq} represent a transition when $1 \leq k \leq n$ and when T contains a triple of the form (q', a, q). The clauses, F, are the following, for $1 \leq k \leq n$: (i) $(\bar{t}_{kaq} \vee x_k^a) \wedge (\bar{t}_{kaq} \vee q_k)$, where x_k^0 denotes $\bar{x}_k$ and x_k^1 denotes x_k; (ii) $(\bar{q}_{k-1} \vee \bigvee\{t_{kaq'} \mid (q,a,q') \in T\})$, for $q \in Q$; (iii) $(\bar{q}_k \vee \bigvee\{t_{kaq} \mid (q',a,q) \in T\})$; (iv) $(\bar{x}_k^a \vee \bigvee\{t_{kaq} \mid (q',a,q) \in T\})$; (v) $(\bar{t}_{kaq'} \vee \bigvee\{q_{k-1} \mid (q,a,q') \in T\})$; together with (vi) $(\bar{q}_0)$ for $q \in Q \setminus I$ and $(\bar{q}_n)$ for $q \in Q \setminus O$.

It is clear that if $F \vdash_1 \bar{x}_k^a$, no string $x_1 \ldots x_n \in L$ can have $x_k = a$. Conversely, assume that $F \not\vdash_1 \bar{x}_k^a$, and in particular that $F \not\vdash_1 \epsilon$. To prove the forcing property, we want to show that some string of L has $x_k = a$. It will be convenient to say that a literal l is 'n.f.' (not falsified) if $F \not\vdash_1 \bar{l}$; thus x_k^a is assumed to be n.f.

By (iv), there's a $(q', a, q) \in T$ such that t_{kaq} is n.f. Hence q_k is n.f., by (i). If $k = n$ we have $q \in O$ by (vi); otherwise some $t_{(k+1)bq'}$ is n.f., by (ii), hence x_{k+1}^b is n.f. Moreover, (v) tells us that there's $(q'', a, q) \in T$ with q''_{k-1} n.f. If $k = 1$ we have $q'' \in I$; otherwise some $t_{(k-1)cq''}$ is n.f., by (iii), and x_{k-1}^c is n.f. Continuing this line of reasoning yields $x_1 \ldots x_n \in L$ with $x_k = a$ (and with $x_{k+1} = b$ if $k < n$, $x_{k-1} = c$ if $k > 1$).

The same proof holds even if we add unit clauses to F that assign values to one or more of the x's. Hence F is forcing. [See F. Bacchus, *LNCS* **4741** (2007), 133–147.]

The language L_2 of exercise 434 yields $17n+4$ clauses: $F = \bigwedge_{1 \leq k \leq n}((\bar{t}_{k00} \vee \bar{x}_k) \wedge (\bar{t}_{k00} \vee 0_k) \wedge (\bar{t}_{k11} \vee x_k) \wedge (\bar{t}_{k11} \vee 1_k) \wedge (\bar{t}_{k12} \vee x_k) \wedge (\bar{t}_{k12} \vee 2_k) \wedge (\bar{t}_{k02} \vee \bar{\bar{x}}_k) \wedge (\bar{t}_{k02} \vee 2_k) \wedge (\bar{0}_{k-1} \vee t_{k00} \vee t_{k11}) \wedge (\bar{1}_{k-1} \vee t_{k12}) \wedge (\bar{2}_{k-1} \vee t_{k02}) \wedge (x_k \vee t_{k00} \vee t_{k02}) \wedge (\bar{x}_k \vee t_{k11} \vee t_{k12}) \wedge (\bar{t}_{k00} \vee 0_{k-1}) \wedge (\bar{t}_{k11} \vee 0_{k-1}) \wedge (\bar{t}_{k12} \vee 1_{k-1}) \wedge (\bar{t}_{k02} \vee 2_{k-1})) \wedge (\bar{1}_0) \wedge (\bar{2}_0) \wedge (\bar{0}_n) \wedge (\bar{1}_n)$. (Unit propagation will immediately assign values to 10 of the $8n+3$ variables, thereby satisfying 22 of these clauses, when $n \geq 3$. For example, $\bar{t}_{112}$, $\bar{t}_{n11}$, $\bar{0}_{n-1}$ are forced.)

The clauses produced by this general-purpose construction can often be significantly simplified by preprocessing to eliminate auxiliary variables. (See exercise 426.)

437. Each variable x_k now becomes a set of $|A|$ variables x_{ka} for $a \in A$, with clauses like (15) and (17) to ensure that exactly one value is assigned. The same construction is then valid, with the same proof, if we simply replace 'x_k^a' by 'x_{ka}' throughout. (Notice that unit propagation will often derive partial information such as $\bar{x}_{ka}$, meaning that $x_k \neq a$, although the precise value of x_k may not be known.)

438. Let $l_{\le j} = l_1 + \cdots + l_j$. Exercise 436 does the job via the following automaton: $Q = \{0, 1, \ldots, l_{\le t} + t - 1\}$, $I = \{0\}$, $O = \{l_{\le t} + t - 1\}$; $T = \{(l_{\le j} + j,\ 0,\ l_{\le j} + j) \mid 0 \le j < t\} \cup \{(l_{\le j} + j + k,\ 1,\ l_{\le j} + j + k + 1) \mid 0 \le j < t,\ 0 \le k < l_{j+1}\} \cup \{(l_{\le j} + j - 1,\ 0,\ l_{\le j} + j - [j = t]) \mid 1 \le j \le t\}$.

439. We obviously want the clauses $(\bar{x}_j \vee \bar{x}_{j+1})$ for $1 \le j < n$; and we can use, say, (18) and (19) with $r = t$, to force 0s whenever the number of 1s reaches t. The difficult part is to force 1s from partial patterns of 0s; for example, if $n = 9$ and $t = 4$, we can conclude that $x_4 = x_6 = 1$ as soon as we know that $x_3 = x_7 = 0$.

An interesting modification of (18) and (19) turns out to work beautifully, namely with the clauses $(\bar{t}_j^k \vee t_{j+1}^k)$ for $1 \le j < 2t - 1$ and $1 \le k \le n - 2t + 1$, together with $(x_{2j+k-1} \vee \bar{t}_{2j-1}^k \vee t_{2j-1}^{k+1})$ for $1 \le j \le t$ and $0 \le k \le n - 2t + 1$, omitting $\bar{t}_{2j-1}^0$ and t_{2j-1}^{n-2t+2}.

440. It's convenient to introduce $\binom{n+1}{2}|N|$ variables P_{ik} for all $P \in N$ and for $1 \le i \le k \le n$, as well as $\binom{n+1}{3}|N|^2$ variables QR_{ijk} for $Q, R \in N$ and for $1 \le i < j \le k \le n$, although almost all of them will be eliminated by unit propagation. The clauses are: (i) $(\overline{QR}_{ijk} \vee Q_{i(j-1)}) \wedge (\overline{QR}_{ijk} \vee R_{jk})$; (ii) $(\overline{P}_{kk} \vee \bigvee\{x_k^a \mid P \to a \in U\})$; (iii) $(\overline{P}_{ik} \vee \bigvee\{QR_{ijk} \mid i < j \le k,\ P \to QR \in W\})$, if $i < k$; (iv) $(\bar{x}_k^a \vee \bigvee\{P_{kk} \mid P \to a \in U\})$; (v) $(\overline{P}_{ik} \vee \bigvee\{PR_{i(k+1)l} \mid k < l \le n, R \in N\} \vee \bigvee\{QR_{hik} \mid 1 \le h < i, Q \in N\})$, if $i > 1$ or $k < n$; (vi) $(\overline{QR}_{ijk} \vee \bigvee\{P_{ik} \mid P \to QR \in W\})$; (vii) $(\overline{P}_{1n})$ for $P \in N \setminus S$.

The forcing property is proved by extending the argument in answer 436: Assume that x_k^a is n.f.; then some P_{kk} with $P \to a$ is also n.f. Whenever P_{ik} is n.f. with $i > 1$ or $k < n$, some $PR_{i(k+1)l}$ or QR_{hik} is n.f.; hence some "larger" P'_{il} or P'_{hk} is also n.f. And if P_{1n} is n.f., we have $P \in S$.

Furthermore we can go "downward": Whenever P_{ik} is n.f. with $i < k$, there's QR_{ijk} such that $Q_{i(j-1)}$ and R_{jk} are n.f.; on the other hand if P_{kk} is n.f., there's $a \in A$ such that x_k^a is n.f. Our assumption that x_k^a is n.f. has therefore shown the existence of $x_1 \ldots x_n \in L$ with $x_k = a$.

[See C.-G. Quimper and T. Walsh, *LNCS* **4741** (2007), 590–604].

441. See O. Bailleux, Y. Boufkhad, and O. Roussel, *LNCS* **5584** (2009), 181–194.

442. (a) $F \,|\, L_q^- = F \,|\, l_1 \,|\, \ldots \,|\, l_{q-1} \,|\, \bar{l}_q$ contains ϵ if and only if $F \,|\, l_1 \,|\, \ldots \,|\, l_{q-1}$ contains ϵ or the unit clause (l_q).

(b) If $F \not\vdash_1 l$ and $F \,|\, \bar{l} \vdash_1 \epsilon$, the failed literal elimination technique will reduce F to $F \,|\, l$ and continue looking for further reductions. Thus we have $F \vdash_2 l$ if and only if unit propagation plus failed literal elimination will deduce either ϵ or l.

(c) Use induction on k; both statements are obvious when $k = 0$. Suppose we have $F \vdash_{k+1} \bar{l}$ via $l_1, \ldots, l_p = \bar{l}$, with $F \,|\, L_q^- \vdash_k \epsilon$ for $1 \le q \le p$. If $p > 1$ we have $F \,|\, l \,|\, L_q^- \vdash_k \epsilon$ for $1 \le q < p$; it follows that $F \,|\, l \vdash_{k+1} l_{p-1}$ and $F \,|\, l \vdash_{k+1} \bar{l}_{p-1}$. If $p = 1$ we have $F \,|\, l \vdash_k \epsilon$. Hence $F \,|\, l \vdash_{k+1} \epsilon$ in both cases.

Now we want to prove that $F \,|\, l \vdash_{k+1} \epsilon$ and $F \vdash_{k+2} \epsilon$, given $F \vdash_{k+1} l'$ and $F \vdash_{k+1} \bar{l}'$. If $F \,|\, L_q^- \vdash_k \epsilon$ for $1 \le q \le p$, with $l_p = l'$, we know that $F \,|\, L_q^- \vdash_{k+1} \epsilon$. Furthermore we can assume that $F \not\vdash_{k+1} \bar{l}$; hence $l \neq \bar{l}_q$ for $1 \le q \le p$, and $l \neq l_p$. If

preprocessing
eliminate
automaton
unit propagation
Quimper
Walsh
Bailleux
Boufkhad
Roussel

$l = l_q$ for some $q < p$, then $F \,|\, l \,|\, L_r^- \vdash_k \epsilon$ for $1 \le r < q$ and $F \,|\, L_r^- \vdash_k \epsilon$ for $q < r \le p$; otherwise $F \,|\, l \,|\, L_q^- \vdash_k \epsilon$ for $1 \le q \le p$. In both cases $F \,|\, l \vdash_{k+1} l'$ and $F \vdash_{k+2} l'$. Essentially the same proof shows that $F \,|\, l \vdash_{k+1} \bar{l}'$ and $F \vdash_{k+2} \bar{l}'$.

(d) True, by the last relation in part (c).

(e) If all clauses of F have more than k literals, $L_k(F)$ is empty; hence $L_0(R') = L_1(R') = L_2(R') = \emptyset$. But $L_k(R') = \{\bar{1}, 2, 4\}$ for $k \ge 3$; for example, $R' \vdash_3 \bar{1}$ because $R' \,|\, 1 \vdash_2 \epsilon$, because $R' \,|\, 1 \vdash_2 3$ and $R' \,|\, 1 \vdash_2 \bar{3}$.

(f) Unit propagation can be done in $O(N)$ steps if N is the total length of all clauses; this handles the case $k = 1$.

For $k \ge 2$, procedure $P_k(F)$ calls $P_{k-1}(F \,|\, x_1)$, $P_{k-1}(F \,|\, \bar{x}_1)$, $P_{k-1}(F \,|\, x_2)$, etc., until either finding $P_{k-1}(F \,|\, \bar{l}) = \{\epsilon\}$ or trying both literals for each variable of F. In the latter case, P_k returns F. In the former case, if $P_{k-1}(F \,|\, l)$ is *also* $\{\epsilon\}$, P_k returns $\{\epsilon\}$; otherwise it returns $P_k(F \,|\, l)$. The set L_k contains all literals for which we've reduced F to $F \,|\, l$, unless $P_k(F) = \{\epsilon\}$. (In the latter case, *every* literal is in L_k.)

To justify this procedure we must verify that the order of testing literals doesn't matter. If $F \,|\, \bar{l} \vdash_k \epsilon$ and $F \,|\, \bar{l}' \vdash_k \epsilon$, we have $F \,|\, l \,|\, \bar{l}' \vdash_k \epsilon$ and $F \,|\, l' \,|\, \bar{l} \vdash_k \epsilon$ by (c); hence $P_k(F \,|\, l) = P_k(F \,|\, l \,|\, l') = P_k(F \,|\, l' \,|\, l) = P_k(F \,|\, l')$.

[See O. Kullmann, *Annals of Math. and Artificial Intell.* **40** (2004), 303–352.]

443. (a) If $F \,|\, L \vdash \epsilon$ then $F \,|\, L \vdash l$ for all literals l; so if $F \in \mathrm{PC}_k$ we have $F \,|\, L \vdash_k l$ and $F \,|\, L \vdash_k \bar{l}$ and $F \,|\, L \vdash_k \epsilon$, proving that $\mathrm{PC}_k \subseteq \mathrm{UC}_k$.

Suppose $F \in \mathrm{UC}_k$ and $F \,|\, L \vdash l$. Then $F \,|\, L \,|\, \bar{l} \vdash \epsilon$, and we have $F \,|\, L \,|\, \bar{l} \vdash_k \epsilon$. Consequently $F \,|\, L \vdash_{k+1} l$, proving that $\mathrm{UC}_k \subseteq \mathrm{PC}_{k+1}$.

The satisfiable clause sets $\emptyset$, $\{1\}$, $\{1, \bar{1}2\}$, $\{12, \bar{1}2\}$, $\{12, \bar{1}2, 1\bar{2}, \bar{1}\bar{2}3\}$, $\{123, \bar{1}23, 1\bar{2}3, \bar{1}\bar{2}3\}$, $\{123, \bar{1}23, 1\bar{2}3, \bar{1}\bar{2}3, 12\bar{3}, \bar{1}2\bar{3}, 1\bar{2}\bar{3}, \bar{1}\bar{2}\bar{3}4\}$, $\ldots$, show that $\mathrm{PC}_k \ne \mathrm{UC}_k \ne \mathrm{PC}_{k+1}$.

(b) $F \in \mathrm{PC}_0$ if and only if $F = \emptyset$ or $\epsilon \in F$. (This can be proved by induction on the number of variables in F, because $\epsilon \notin F$ implies that F has no unit clauses.)

(c) If F has only one clause, it is in UC_0. More interesting examples are $\{1\bar{2}, \bar{1}2\}$; $\{1234, \bar{1}\bar{2}\bar{3}\bar{4}\}$; $\{123\bar{4}, 12\bar{3}4, 1\bar{2}34, \bar{1}234\}$; $\{12, \bar{1}\bar{2}, 34\bar{5}, \bar{3}45\}$; etc. In general, F is in UC_0 if and only if it contains all of its prime clauses.

(d) True, by induction on n: If $F \,|\, L \vdash l$ then $F \,|\, L \,|\, \bar{l} \vdash \epsilon$, and $F \,|\, L \,|\, \bar{l}$ has $\le n-1$ variables; so $F \,|\, L \,|\, \bar{l} \in \mathrm{PC}_{n-1} \subseteq \mathrm{UC}_{n-1}$. Hence we have $F \,|\, L \,|\, \bar{l} \vdash_{n-1} \epsilon$ and $F \,|\, L \vdash_n l$.

(e) False, by the examples in (c).

(f) $R' \in \mathrm{UC}_2 \setminus \mathrm{PC}_2$. For example, we have $R' \,|\, 1 \vdash_2 2$ and $R' \,|\, 1 \vdash_2 \bar{2}$.

[See M. Gwynne and O. Kullmann, `arXiv:1406.7398` [cs.CC] (2014), 67 pages.]

444. (a) Complementing a variable doesn't affect the algorithm's behavior, so we can assume that F consists of unrenamed Horn clauses. Then all clauses of F will be Horn clauses of length ≥ 2 whenever step E2 is reached. Such clauses are always satisfiable, by setting all remaining variables false; so step E3 cannot find both $F \vdash_1 l$ and $F \vdash_1 \bar{l}$.

(b) For example, $\{12, \bar{2}3, 1\bar{2}\bar{3}, \bar{1}23\}$.

(c) Every unsatisfiable F recognized by SLUR must be in UC_1. Conversely, if $F \in \mathrm{UC}_1$, we can prove that F is satisfiable and in UC_1 whenever step E2 is reached.

[Essentially the same argument proves that a generalized algorithm, which uses $\vdash_k$ instead of $\vdash_1$ in steps E1 and E3, always classifies F if and only if $F \in \mathrm{UC}_k$. See M. Gwynne and O. Kullmann, *Journal of Automated Reasoning* **52** (2014), 31–65.]

(d) If step E3 interleaves unit propagation on $F \,|\, l$ with unit propagation on $F \,|\, \bar{l}$, stopping when either branch is complete and ϵ was not detected in the other, the running time is proportional to the number of cells used to store F, using data structures like those of Algorithm L. (This is an unpublished idea of Klaus Truemper.)

Kullmann
prime clauses
Gwynne
Kullmann
propagation, kth order
UC_k
Gwynne
Kullmann
data structures
Truemper

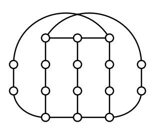
[SLUR is due to J. S. Schlipf, F. S. Annexstein, J. V. Franco, and R. P. Swaminathan, *Information Processing Letters* **54** (1995), 133–137.]

445. (a) Since the lexicographic constraints (169) are forcing, a succinct certificate is $(\bar{x}_{1m}, \bar{x}_{2m}, \ldots, \bar{x}_{(m-1)m}, \bar{x}_{2(m-1)}, \bar{x}_{3(m-1)}, \ldots, \bar{x}_{(m-1)(m-1)}, \bar{x}_{3(m-2)}, \bar{x}_{4(m-2)}, \ldots, \bar{x}_{(m-1)(m-2)}, \ldots, \bar{x}_{(m-1)2}, \emptyset)$. The first $m-1$ steps can be replaced by 'x_{0m}'.

(b) $(\bar{x}_{(m-1)1}, \bar{x}_{(m-2)2}, \ldots, \bar{x}_{1(m-1)}, \emptyset)$.

(c) $(x_{01}, x_{12}, \ldots, x_{(m-2)(m-1)}, \emptyset)$.

446. $Z(m,n)-1$, because a 4-cycle corresponds to a quad.



447. For general m and n we can add the $m^{\underline{3}}n^{\underline{3}}/3!$ constraints $(\bar{x}_{ij} \vee \bar{x}_{i'j} \vee \bar{x}_{i'j'} \vee \bar{x}_{i''j'} \vee \bar{x}_{i''j''} \vee \bar{x}_{ij''})$ to (184), for $1 \le i < i' < i'' \le m$ and distinct $\{j, j', j''\} \subseteq \{1, \ldots, n\}$. The 19-edge graph illustrated here works when $m = n = 8$; and Algorithm C finds girth ≥ 8 unsatisfiable with 20 edges, after only 400 megamems of calculation (using lexicographic row/column symmetry).

448. Each pair of points can occur together in at most one line. If the lines contain respectively $l_1, \ldots, l_n$ points, we therefore have $\binom{l_1}{2} + \cdots + \binom{l_n}{2} \le \binom{m}{2} = 3n$. A Steiner triple system achieves equality, with $l_1 = \cdots = l_n = 3$. Since $\binom{l-1}{2} + \binom{l'+1}{2} < \binom{l}{2} + \binom{l'}{2}$ when $l \ge l' + 2$, we can't have $l_1 + \cdots + l_n > 3n$. Thus $Z(m,n) = 3n+1$.

[If m is even and $\binom{m}{2} = 3n$, we can't cover all the pairs with triples, because no point can be in more than $(m-2)/2$ triples. Daniel Horsley has shown that $Z(m,n) = 3n + \lfloor 1 - m/14 \rfloor$ in such cases; see (a paper in preparation).]

449. It's wise to try first for symmetric solutions with $x_{ij} = x_{ji}$, roughly halving the number of variables; then the matrices below are found quickly. Such solutions are impossible when n = 9, 12, 13 (and also when n = 15 and 16 if we insist on five 1s in the top row). The case $n = 13$ corresponds to the projective plane of order 3; indeed, a projective plane of order q is equivalent to a maximum quad-free matrix with $m = n = q^2 + q + 1$ and $Z(n,n) = (q+1)n + 1$.

								1111100000000000
							111110000000000	1000111000000000
						11110000000000	100011100000000	1000000111100000
					1111000000000	10001110000000	100000011100000	1000000000011100
				111100000000	1000111000000	10000001110000	100000000011110	0100100100010000
			11110000000	100011100000	1000000111000	10000000001110	010010010010000	0100010010001010
		1111000000	10001100000	100000011000	1000000000111	01000100001001	010001001001000	0100000001000101
	111100000	1000110000	10000011100	100000000111	0100100100100	01001001000100	010000000100101	0010100010000100
11100000	100011100	1000001100	10000000011	010010010100	0100010010010	01000010100010	001010001000011	0010010100000001
10011000	100000011	1000000011	01001010010	010001001010	0100001001001	00100101000010	001001010000100	0010001001001000
10000110	010010010	0100101010	01000001001	010000100001	0010100010001	00100010011000	001000100110000	0010000000110010
01010100	010001001	0100000101	00101001000	001010001001	0010010001100	00100000100101	000110000101000	0001100000101001
01000011	001010001	0010100001	00100110001	001000110010	0010001100010	00011000101000	000101000010001	0001010001010000
00110001	001000110	0010010110	00100000110	000110000010	0001100001010	00010100010100	000100101000100	0001001100000110
00101010	000101010	0001100100	00011000101	000101010001	0001010100001	00010011000001	000100010000010	0000010000100100
00001101	000100101	0001011001	00010101010	000100101100	0001001010100	00001000010011	000000110001001	0000001010010001

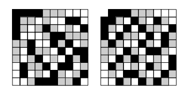
450. To prove the hint, add the unary clause $(\bar{x}_{15})$ to the others; this problem is rapidly found to be unsatisfiable, hence no line has more than 4 points. On the other hand, a line with fewer than 3 points is impossible because $Z(9,10) = 32$. The same arguments show that every point belongs to either 3 or 4 lines. Thus exactly four lines contain four points, and exactly four points lie on such lines.

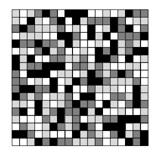
If $p \in l$ and l is a 4-point line, every other line containing p must contain 2 of the remaining 6 points. And the four 4-point lines contain at least $4 \times 4 - \binom{4}{2} = 10$ points altogether. Hence, pigeonwise, we see that each of the four 4-point lines contains exactly one of the four 4-line points.

Now we may call the 4-line points $\{a, b, c, d\}$, and the 4-point lines $\{A, B, C, D\}$. The other points may be called $\{ab, ac, ad, bc, bd, cd\}$, with $A = \{a, ab, ac, ad\}$, $B = \{b, ab, bc, bd\}$, $C = \{c, ac, bc, cd\}$, $D = \{d, ad, bd, cd\}$. The other lines can be called $\{AB, AC, AD, BC, BD, CD\}$; and we have $AB = \{a, b, cd\}$, $AC = \{a, c, ad\}$, etc.

Schlipf
Annexstein
Franco
Swaminathan
forcing
4-cycle
quad
lexicographic row/column symmetry
Steiner triple system
Horsley
symmetric solutions
projective plane

451. One of the colors can be placed uniquely, by the previous exercise. So we're left with the simple problem of two-coloring the remaining 66 squares and avoiding both 0-quads and 1-quads. That problem is unsatisfiable with $\sum x_{ij}$ odd. The author then constructed a $33+33+33$ solution by hand, using the fact that each color class must be unable to use the deleted square. [See M. Beresin, E. Levine, and J. Winn, *The College Mathematics Journal* **20** (1989), 106–114 and the cover; J. L. Lewis, *J. Recreational Math.* **28** (1997), 266–273.]





452. Any such solution must have exactly 81 cells of each color, because R. Nowakowski proved in 1978 that $Z(18,18)=82$. The solution exhibited here was found by B. Steinbach and C. Posthoff [*Multiple-Valued Logic and Soft Computing* **21** (2013), 609–625], exploiting 90° rotational symmetry.

453. (a) If $R\subseteq\{1,\ldots,m\}$ and $C\subseteq\{1,\ldots,n\}$, let $V(R,C)=\{u_i\mid i\in R\}\cup\{v_j\mid j\in C\}$. If X is decomposable, there's no path from a vertex in $V(R,C)$ to a vertex not in $V(R,C)$; hence the graph isn't connected. Conversely, if the graph isn't connected, let $V(R,C)$ be one of its connected components. Then $0<|R|+|C|<m+n$, and we've decomposed X.

(b) False in general, unless every row and column of X' contains a positive element. Otherwise, clearly true by the definition of lexicographic order.

(c) True: A direct sum is certainly decomposable. Conversely, let X be decomposable via R and C. We may assume that $1\in R$ or $1\in C$; otherwise we could replace R by $\{1,\ldots,m\}\setminus R$ and C by $\{1,\ldots,n\}\setminus C$. Let $i\ge 1$ and $j\ge 1$ be minimal such that $i\notin R$ and $j\notin C$. Then $x_{i'j}=0$ for $1\le i'<i$ and $x_{ij'}=0$ for $1\le j'<j$. The lexicographic constraints now force $x_{i'j'}=0$ for $1\le i'<i$, $j'\ge j$; also for $i'\ge i$, $1\le j'<j$. Consequently $X=X'\oplus X''$, where X' is $(i-1)\times(j-1)$ and X'' is $(m+1-i)\times(n+1-j)$. (Degenerate cases where $i=1$ or $j=1$ or $i=m+1$ or $j=n+1$ need to be considered, but they work fine. This result allows us to "read off" the block decomposition of a lexicographically ordered matrix.)

Reference: A. Mader and O. Mutzbauer, *Ars Combinatoria* **61** (2001), 81–95.

454. We have $f(x)\le f(x\tau)\le f(x\tau\tau)\le\cdots\le f(x\tau^k)\le\cdots$; eventually $x\tau^k=x$.

455. (a) Yes, because C only causes 1001 and 1101 to be nonsolutions. (b) No, because F might have been satisfied only by 0011. (c) Yes as in (a), although (187) might no longer be an endomorphism of $F\wedge C$ as it was in that case. (d) Yes; if 0110 is a solution, so are 0101 and 1010. [Of course this exercise is highly artificial: We're unlikely to know that a weird mapping such as (187) is an endomorphism of F unless we know a lot more about the set of solutions.]

456. Only $(1+2\cdot 7)(1+2)(1+8)=405$, out of 65536 possibilities (about 0.06%).

457. We have $\min_{0\le k\le 16}(k^k 16^{16-k})=6^6 16^{10}\approx 51.3\times 10^{16}$. For general n, the minimum occurs when $k=2^n/e+O(1)$; and it is $2^{2^n(n-x)}$ where $x=1/(e\ln 2)+O(2^{-n})<1$.

458. The operation of assigning values to each variable of an autarky, so that all clauses containing those variables are satisfied, while leaving all other variables unchanged, is an endomorphism. (For example, consider the operation that makes a pure literal true.)

459. $\mathrm{sweep}(X_{ij})=0$ when $i=0$ or $j=0$. And for $1\le i\le m$ and $1\le j\le n$ we have $\mathrm{sweep}(X_{ij})=\max\bigl(x_{ij}+\mathrm{sweep}(X_{(i-1)(j-1)}),\mathrm{sweep}(X_{(i-1)j}),\mathrm{sweep}(X_{i(j-1)})\bigr)$.

[Let the 1s in the matrix be $x_{i_1j_1},\ldots,x_{i_rj_r}$, with $1\le i_1\le\cdots\le i_r\le m$ and with $j_{q+1}<j_q$ when $i_{q+1}=i_q$. Richard Stanley has observed (unpublished) that $\mathrm{sweep}(X)$ is the number of rows that occur when the Robinson–Schensted–Knuth algorithm is used to insert the sequence $n-j_1,\ldots,n-j_r$ into an initially empty tableau.]

Knuth
Beresin
Levine
Winn
Lewis
Nowakowski
Steinbach
Posthoff
rotational symmetry
block decomposition
Mader
Mutzbauer
pure literal
Stanley
Robinson
Schensted
Knuth
tableau

Durfee square
disjoint shortest paths
unit propagation

460. We introduce auxiliary variables s_{ij}^t that will become true if $\mathrm{sweep}(X_{ij}) > t$. They are implicitly true when $t < 0$, false when $t = k$. The clauses are as follows, for $1 \le i \le m$, $1 \le j \le n$, and $0 \le t \le \min(i-1, j-1, k)$: $(\bar{s}_{(i-1)j}^t \vee s_{ij}^t)$, if $i > 1$ and $t < k$; $(\bar{s}_{i(j-1)}^t \vee s_{ij}^t)$, if $j > 1$ and $t < k$; and $(\bar{x}_{ij} \vee \bar{s}_{(i-1)(j-1)}^{t-1} \vee s_{ij}^t)$. Omit $\bar{s}_{0(j-1)}^{t-1}$ and $\bar{s}_{(i-1)0}^{t-1}$ and $\bar{s}_{(i-1)(j-1)}^0$ and $\bar{s}_{ij}^k$ from that last clause, if present.

461. $\bigvee_{i=1}^{m-1} \bigvee_{j=1}^{n-1} (x_{ij} \vee \bar{c}_{(i-1)j} \vee c_{ij}) \wedge \bigvee_{i=1}^{m} \bigvee_{j=1}^{n-1} (\bar{c}_{(i-1)j} \vee \bar{x}_{ij} \vee x_{i(j+1)})$, omitting $\bar{c}_{0j}$. These clauses take care of τ_1; interchange $i \leftrightarrow j$, $m \leftrightarrow n$ for τ_2.

462. Let $\widetilde{X}_{ij}$ denote the *last* $m+1-i$ rows and the last $n+1-j$ columns of X; and let $t_{ij} = \mathrm{sweep}(X_{(i-1)(j-1)}) + \mathrm{sweep}(\widetilde{X}_{(i+1)(j+1)})$. For τ_1 we must prove $1 + t_{i(j+1)} \le k$, given that $1 + t_{ij} \le k$. It's true because $\mathrm{sweep}(X_{(i-1)j}) = \mathrm{sweep}(X_{(i-1)(j-1)})$ when column j begins with $i-1$ zeros, and we have $\mathrm{sweep}(\widetilde{X}_{(i+1)(j+2)}) \le \mathrm{sweep}(\widetilde{X}_{(i+1)(j+1)})$.

Let $X' = X\tau_3$ have the associated sweep sums t'_{ij}. We must prove that $t'_{ij} \le k$ and $1 + t'_{(i+1)(j+1)} \le k$, if $1 + t_{ij} \le k$, $1 + t_{i(j+1)} \le k$, $1 + t_{(i+1)j} \le k$, and $t_{(i+1)(j+1)} \le k$. The key point is that $\mathrm{sweep}(X'_{ij}) = \max(\mathrm{sweep}(X_{(i-1)j}), \mathrm{sweep}(X_{i(j-1)}))$, since $x'_{ij} = 0$. Also $\mathrm{sweep}(\widetilde{X}'_{(i+1)(j+1)}) = 1 + \mathrm{sweep}(\widetilde{X}_{(i+2)(j+1)})$.

(Notice that τ_1 and τ_2 might actually *decrease* the sweep, but τ_3 preserves it.)

463. If row $i+1$ is entirely zero but row i isn't, τ_2 will apply. Therefore the all-zero rows occur at the top. And by τ_1, the first nonzero row has all its 1s at the right.

Suppose rows 1 through i have $r_1, \ldots, r_i$ 1s, all at the right, with $r_i > 0$. Then $r_1 \le \cdots \le r_i$, by τ_2. If $i < n$ we can increase i to $i+1$, since we can't have $x_{(i+1)j} > x_{(i+1)(j+1)}$ when $j \le n - r_i$, by τ_1; and we can't have it when $j > n - r_i$, by τ_3.

Thus all the 1s are clustered at the right and the bottom, like the diagram of a partition but rotated 180°; and the sweep is the size of its "Durfee square" (see Fig. 48 in Section 7.2.1.4). Hence the maximum number of 1s, given sweep k, is $k(m+n-k)$.

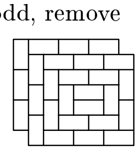
464. By answer 462, τ_1 can be strengthened to τ'_1, which sets $x_{i(j+1)} \leftarrow 1$ but leaves $x_{ij} = 1$. Similarly, τ_2 can be strengthened to τ'_2. These endomorphisms preserve the sweep but increase the weight, so they can't apply to a matrix of maximum weight. [One can prove, in fact, that max-weight binary matrices of sweep k are precisely equivalent to k disjoint shortest paths from the leftmost cells in row m to the rightmost cells in row 1. Hence every integer matrix of sweep k is the sum of k matrices of sweep 1.]

465. If not, there's a cycle $x_0 \to x_1 \to \cdots \to x_p = x_0$ of length $p > 1$, where $x_i \tau_{uv_i} \mapsto x_{i+1}$. Let uv be the largest of $\{uv_1, \ldots, uv_{p-1}\}$. Then none of the other τ's in the cycle can change the status of edge uv. But that edge must change status at least twice.

466. Notice first that v_{11} must be true, if $m \ge 2$. Otherwise h_{11}, v_{21}, h_{22}, v_{32}, ... would successively be forced by unit propagation, until reaching a contradiction at the edge of the board. And v_{31} must also be true, if $m \ge 4$, by a similar argument. Thus the entire first column must be filled with verticals, except the bottom row when m is odd.

Then we can show that the remainder of row 1 is filled with horizontals, except for the rightmost column when n is even. And so on.

The unique solution when m and n are both even uses v_{ij} if and only if $i+j$ is even and $\max(i, m-i) \le j \le n/2$, or $i+j$ is odd and $v_{i(n+1-j)}$ is used. When m is odd, add a row of horizontals below the $(m-1) \times n$ solution. When n is odd, remove the rightmost column of verticals in the $m \times (n+1)$ solution.



467. The 8×7 covering is obtained by reflection of the 7×8 covering (shown here) about its southwest-to-northeast diagonal. Both solutions are unique.

even-length cycle
truth table
orbits
cycle

468. (a) Typical running times with Algorithm C for sizes 6×6, 8×8, $\ldots$, 16×16 are somewhat improved: $39\,\mathrm{K}\mu$, $368\,\mathrm{K}\mu$, $4.3\,\mathrm{M}\mu$, $48\,\mathrm{M}\mu$, $626\,\mathrm{M}\mu$, $8\,\mathrm{G}\mu$.

(b) Now they're even better, but still growing exponentially: $30\,\mathrm{K}\mu$, $289\,\mathrm{K}\mu$, $2.3\,\mathrm{M}\mu$, $22\,\mathrm{M}\mu$, $276\,\mathrm{M}\mu$, $1.7\,\mathrm{G}\mu$.

469. For instance (v_{11}), (v_{31}), (v_{51}), (h_{12}), (h_{14}), (v_{22}), (v_{42}), (h_{23}), (v_{33}), ϵ.

470. There can't be a cycle $x_0 \to x_1 \to \cdots \to x_p = x_0$ of length $p > 1$, because the largest vertex whose mate is changed always gets smaller and smaller mates.

471. We must pair $2n$ with 1, then $2n-1$ with 2, $\ldots$, then $n+1$ with n.

472. We can number the vertices from 1 to mn in such a way that every 4-cycle switches as desired. For example, we can make $(i,j) < (i,j+1) \iff (i,j) < (i+1,j) \iff (i,j) \bmod 4 \in \{(0,0),\ (0,1),\ (1,1),\ (1,2),\ (2,2),\ (2,3),\ (3,3),\ (3,0)\}$. One such numbering in the 4×4 case is shown here.

16	15	1	2
4	14	13	3
5	6	12	11
9	7	8	10

473. For every even-length cycle $v_0 — v_1 — \cdots — v_{2r-1} — v_0$ with $v_0 = \max v_i$ and $v_1 > v_{2r-1}$, assert $(\overline{v_0v_1} \lor v_1v_2 \lor \overline{v_2v_3} \lor \cdots \lor v_{2r-1}v_0)$.

474. (a) $(2n)\cdot(2n-2)\cdot\ldots\cdot 2 = 2^n n!$. (b) $(17\bar{3})(\bar{1}\bar{7}3)(25\bar{2}\bar{5})(4\bar{4})(6)(\bar{6})$.

(c) Using $\mathtt{0}$, $\mathtt{1}$, $\ldots$, $\mathtt{f}$ for the 4-tuples 0000, 0001, $\ldots$, 1111, we must have $f(\mathtt{0}) = f(\mathtt{9}) = f(\mathtt{5})$, $f(\mathtt{2}) = f(\mathtt{b}) = f(\mathtt{7})$; $f(\mathtt{4}) = f(\mathtt{8}) = f(\mathtt{d})$; and $f(\mathtt{6}) = f(\mathtt{a}) = f(\mathtt{f})$; in other words, the truth table of f must have the form *abcdeagceagcfehg*, where $a,b,c,d,e,f,g,h \in \{0,1\}$. So there are 2^8 f's.

(d) Change '$=$' to '$\neq$' in (c). There are no such truth tables, because (191) contains odd cycles; all cycles of an antisymmetry must have even length.

(e) The 128 binary 7-tuples are partitioned into sixteen "orbits" $\{x, x\sigma, x\sigma^2, \ldots\}$, with eight of size 12 and eight of size 4. For example, one of the 4s is $\{0011010, 0010110, 0111110, 0110010\}$; one of the 12s is $\{0000000, 0011101, \ldots, 1111000\}$. Hence there are 2^{16} functions with this symmetry, and 2^{16} others with this antisymmetry.

475. (a) $2^{n+1}n!$. (There are $2^{n+1}n!/a$, if f has a automorphisms+antiautomorphisms.)

(b) $(x\bar{z})(\bar{x}z)$, because (surprisingly) $(x \lor y) \land (x \oplus z) = (\bar{z} \lor y) \land (\bar{z} \oplus \bar{x})$.

(c) In general if σ is *any* permutation having a cycle of length l, and if p is a prime divisor of l, some power of σ will have a cycle of length p. (Repeatedly raise σ to the qth power for all primes $q \neq p$, until all cycle lengths are powers of p. Then, if the longest remaining cycle has length p^e, compute the p^{e-1}st power.)

(d) Suppose $f(x_1,x_2,x_3)$ has the symmetry $(x_1\bar{x}_2x_3)(\bar{x}_1x_2\bar{x}_3)$. Then $f(0,0,0) = f(1,1,0) = f(0,1,1)$, $f(1,1,1) = f(0,0,1) = f(1,0,0)$, so $(x_1\bar{x}_2)(\bar{x}_1x_2)$ is a symmetry.

(e) A similar argument shows that $(ux)(vw)(\bar{u}\bar{x})(\bar{v}\bar{w})$ is a symmetry.

(f) If σ is an antisymmetry of f, then σ^2 is a symmetry. If f has a nontrivial symmetry, it has a symmetry of prime order p, by (c). And if $p \neq 2$, it has one of order 2, by (d) and (e), unless $n > 5$.

(g) Let $f(x_1,\ldots,x_6) = 1$ only when $x_1\ldots x_6 \in \{001000, 001001, 001011, 010000, 010010, 010110, 100000, 100100, 100101\}$. (Another interesting example, for $n = 7$, has $f = 1 \iff x_1\ldots x_7$ is a cyclic shift of 0000001, 0001101, or 0011101; 21 symmetries.)

476. We want clauses that specify r-step chains in n variables, having a single output x_{n+r}. For $0 < t < t' < 2^n$, introduce new variables $\Delta_{tt'} = x_{(n+r)t} \oplus x_{(n+r)t'}$. (See (24).) Then for each signed involution σ, not the identity, we want a clause that says "σ is not a symmetry of f," namely $(\bigvee\{\Delta_{tt'} \mid t < t' \text{ and } t' = t\sigma\})$. (Here t is considered to be the same as its binary representation $(t_1\ldots t_n)_2$, as in exercise 477.)

Also, if σ has no fixed points — this is true if and only if σ takes $x_i \mapsto \bar{x}_i$ for at least one i — we have further things to do: In case (b), we want a clause that says "σ

is not an antisymmetry," namely $(\bigvee\{\overline{\Delta}_{tt'} \mid t < t' \text{ and } t' = t\sigma\})$. But in case (a), we need further variables a_j for $1 \le j \le T$, where T is the number of signed involutions that are fixedpoint-free. We append the clause $(a_1 \vee \cdots \vee a_T)$, and also $(\bar{a}_j \vee \Delta_{tt'})$ for all $t < t'$ such that $t' = t\sigma$ when σ corresponds to index j. Those clauses say, "there's at least one signed involution that is an antisymmetry."

There are no solutions when $n \le 3$. Answers for (a) are $(((x_1 \oplus x_2) \vee x_3) \wedge x_4) \oplus x_1$ and $((((\bar{x}_1 \oplus x_2) \wedge x_3) \oplus x_4) \wedge x_5) \oplus x_1$; in both cases the signed involution $(1\bar{1})(2\bar{2})$ is obviously an antisymmetry. Answers for (b) are $((x_1 \oplus x_2) \vee x_3) \wedge (x_4 \vee x_1)$ and $(((x_1 \wedge x_2) \oplus x_3) \wedge x_4) \oplus (x_5 \vee x_1)$. [Is there a simple formula that works for all n?]

477. Use the following variables for $1 \le h \le m$, $n < i \le n + r$, and $0 < t < 2^n$: $x_{it} =$ (tth bit of truth table for x_i); $g_{hi} = [g_h = x_i]$; $s_{ijk} = [x_i = x_j \circ_i x_k]$, for $1 \le j < k < i$; $f_{ipq} = \circ_i(p, q)$ for $0 \le p, q \le 1$, $p+q > 0$. (We don't need f_{i00}, because every operation in a normal chain takes $(0,0) \mapsto 0$.) The main clauses for truth table computations are

$$(\bar{s}_{ijk} \vee (x_{it} \oplus a) \vee (x_{jt} \oplus b) \vee (x_{kt} \oplus c) \vee (f_{ibc} \oplus \bar{a})), \quad \text{for } 0 \le a, b, c \le 1 \text{ and } 1 \le j < k < i.$$

Simplifications arise in special cases: For example, if $b = c = 0$, the clause is omitted if $a = 0$, and the term f_{i00} is omitted if $a = 1$. Furthermore if $t = (t_1 \ldots t_n)_2$, and if $j \le n$, the (nonexistent) variable x_{jt} actually has the known value t_j; again we omit either the whole clause or the term $(x_{jt} \oplus b)$, depending on b and t. For example, there usually are eight main clauses that involve s_{ijk}; but there's only one that involves s_{i12} when $t < 2^{n-2}$, namely $(\bar{s}_{i12} \vee \bar{x}_{i1})$, because the truth tables for x_1 and x_2 begin with 2^{n-2} 0s. (All such simplifications would be done by a preprocessor if we had defined additional variables f_{i00} and x_{jt}, and fixed their values with unit clauses.)

There also are more mundane clauses, namely $(\bar{g}_{hi} \vee \bar{x}_{it})$ or $(\bar{g}_{hi} \vee x_{it})$ according as $g_h(t_1, \ldots, t_n) = 0$ or 1, to fix the outputs; also $(\bigvee_{i=n+1}^{n+r} g_{hi})$ and $(\bigvee_{k=1}^{i-1} \bigvee_{j=1}^{k-1} s_{ijk})$, to ensure that each output appears in the chain and that each step has two operands.

Additional clauses are optional, but they greatly shrink the space of possibilities: $(\bigvee_{k=1}^{m} g_{ki} \vee \bigvee_{i'=i+1}^{n+r} \bigvee_{j=1}^{i-1} s_{i'ji} \vee \bigvee_{i'=i+1}^{n+r} \bigvee_{j=i+1}^{i'-1} s_{i'ij})$ ensures that step i is used at least once; $(\bar{s}_{ijk} \vee \bar{s}_{i'ji})$ and $(\bar{s}_{ijk} \vee \bar{s}_{i'ki})$ for $i < i' \le n + r$ avoid reapplying an operand.

Finally, we can rule out trivial binary operations with the clauses $(f_{i01} \vee f_{i10} \vee f_{i11})$, $(f_{i01} \vee \bar{f}_{i10} \vee \bar{f}_{i11})$, $(\bar{f}_{i01} \vee f_{i10} \vee \bar{f}_{i11})$. (But beware: These clauses, for $n < i \le n+r$, will make it impossible to compute the trivial function $g_1 = 0$ in fewer than three steps!)

Further clauses such as $(\bar{s}_{ijk} \vee f_{i01} \vee \bar{x}_{it} \vee x_{jt})$ are true, but unhelpful in practice.

478. We can insist that the (j, k) pairs in steps $n+1, \ldots, n+r$ appear in colexicographic order; for example, a chain step like $x_8 = x_4 \oplus x_5$ need never follow $x_7 = x_2 \wedge x_6$. The clauses, for $n < i < n+r$, are $(\bar{s}_{ijk} \vee \bar{s}_{(i+1)j'k'})$ if $1 \le j' < j < k = k' < i$ or if $1 \le j < k$ and $1 \le j' < k' < k < i$. (If $(j,k) = (j',k')$, we could insist further that $f_{i01}f_{i10}f_{i11}$ is lexicographically less than $f_{(i+1)01}f_{(i+1)10}f_{(i+1)11}$. But the author didn't go that far.)

Furthermore, if $p<q$ and if each output function is unchanged when x_p is swapped with x_q, we can insist that x_p is used before x_q as an operand. Those clauses are

$$(\bar{s}_{ijq} \vee \bigvee_{n<i'<i} \bigvee_{1 \le j' < k' < i'} [j' = p \text{ or } k' = p]\, s_{i'j'k'}) \text{ whenever } j \ne p.$$

For example, when answer 477 is applied to the full-adder problem, it yields M_r clauses in N_r variables, where $(M_4, M_5) = (942, 1662)$ and $(N_4, N_5) = (82, 115)$. The symmetry-breaking strategy above, with $(p,q) = (1,2)$ and $(2,3)$, raises the number of clauses to M'_r, where $(M'_4, M'_5) = (1025, 1860)$. Algorithm C reported 'unsat' after $(1015, 291)$ kilomems using (M_4, M'_4) clauses; 'sat' after $(250, 268)$ kilomems using (M_5, M'_5). With larger problems, such symmetry breakers give significant speedup when proving unsatisfiability, but they're often a handicap in satisfiable instances.

normal chain
preprocessor
colexicographic order
author
full-adder

sideways sum
author
footprint heuristic
normal function

479. (a) Using the notation of the previous answer, we have $(M_8, M_8', N_8) = (14439, 17273, 384)$ and $(M_9, M_9', N_9) = (19719, 24233, 471)$. The running times for the 'sat' cases with M_9 and M_9' clauses were respectively $(16, 645, 1259)$ and $(66, 341, 1789)$ megamems — these stats are the (min, median, max) of nine runs with different random seeds. The 'unsat' cases with M_8 and M_8' were dramatically different: $(655631, 861577, 952218)$ and $(8858, 10908, 13171)$. Thus $s(4) = 9$ in 7.1.2–(28) is optimum.

(b) But $s(5) = 12$ is *not* optimum, despite the beauty of 7.1.2–(29)! The $M_{11} = 76321$ clauses in $N_{11} = 957$ variables are 'sat' in 680 Gμ, yielding an amazing chain:

$$\begin{array}{lll} x_6 = x_1 \oplus x_2, & x_{10} = x_6 \vee x_7, & \\ x_7 = x_1 \oplus x_3, & x_{11} = x_4 \oplus x_9, & x_{14} = \bar{x}_8 \wedge x_{11}, \\ x_8 = x_4 \oplus x_5, & x_{12} = x_9 \oplus x_{10}, & z_1 = x_{15} = x_{10} \oplus x_{14}, \\ x_9 = x_3 \oplus x_6, & z_0 = x_{13} = x_5 \oplus x_{11}, & z_2 = x_{16} = x_{12} \wedge \bar{x}_{15}. \end{array}$$

And $(M_{10}', N_{10}) = (68859, 815)$ turns out to be 'unsat' in 1773 gigamems; this can be reduced to 309 gigamems by appending the unit clause $(g_{3(15)})$, since $C(S_{4,5}) = 10$.

Hence we can evaluate $x_1 + \cdots + x_7$ in only $5 + 11 + 2 + 1 = 19$ steps, by computing $(u_1u_0)_2 = x_5 + x_6 + x_7$, $(v_2v_1z_0)_2 = x_1 + x_2 + x_3 + x_4 + u_0$, $(w_2z_1)_2 = u_1 + v_1$, $z_2 = v_2 \oplus w_2$.

(c) The solver finds an elegant 8-step solution for $(M_8, N_8) = (6068, 276)$ in 6 Mμ:

$$\begin{array}{llll} x_4 = x_1 \vee x_2, & x_6 = x_3 \oplus x_4, & x_8 = x_3 \oplus x_5, & S_1 = x_{10} = x_6 \wedge x_8, \\ x_5 = x_1 \oplus x_2, & \overline{S}_0 = x_7 = x_3 \vee x_4, & S_3 = x_9 = \bar{x}_6 \wedge x_8, & S_2 = x_{11} = x_7 \oplus x_8. \end{array}$$

The corresponding $(M_7', N_7) = (5016, 217)$ problem is 'unsat' in 97 Mμ.

(d) The total cost of evaluating the S's independently is $3 + 7 + 6 + 7 + 3 = 26$, using the optimum computations of Fig. 9 in Section 7.1.2. Therefore the author was surprised to discover a 9-step chain for S_1, S_2, and S_3, using the footprint heuristic:

$$\begin{array}{lll} x_5 = x_1 \oplus x_2, & x_8 = x_5 \oplus x_7, & S_3 = x_{11} = \bar{x}_8 \wedge x_9, \\ x_6 = x_1 \oplus x_3, & x_9 = x_6 \vee x_7, & S_2 = x_{12} = x_8 \wedge \bar{x}_{10}, \\ x_7 = x_3 \oplus x_4, & x_{10} = x_2 \oplus x_9, & S_1 = x_{13} = x_8 \wedge x_{10}. \end{array}$$

This chain can solve problem (d) in 13 steps; but SAT technology does it in 12(!):

$$\begin{array}{lll} x_5 = x_1 \oplus x_2, & x_9 = x_6 \vee x_7, & S_1 = x_{13} = x_8 \wedge x_{10}, \\ x_6 = x_1 \oplus x_3, & x_{10} = x_2 \oplus x_9, & S_4 = x_{14} = x_1 \wedge \bar{x}_{11}, \\ x_7 = x_3 \oplus x_4, & x_{11} = x_5 \vee x_9, & \overline{S}_0 = x_{15} = x_4 \vee x_{11}, \\ x_8 = x_5 \oplus x_7, & S_3 = x_{12} = x_8 \wedge \bar{x}_{10}, & S_2 = x_{16} = \bar{x}_8 \wedge x_{11}. \end{array}$$

The nonexistence of an 11-step solution can be proved via Algorithm C by a long computation (11034 gigamems), during which 99,999,379 clauses are learned(!).

(e) This solution (found in 342 Gμ) matches the lower bound in exercise 7.1.2–80:

$$\begin{array}{lll} x_7 = x_1 \oplus x_2, & x_{11} = x_4 \oplus x_{10}, & \\ x_8 = x_3 \oplus x_4, & x_{12} = x_5 \oplus x_{10}, & x_{15} = \bar{x}_9 \wedge x_{12}, \\ x_9 = x_1 \oplus x_5, & x_{13} = x_8 \vee x_{11}, & x_{16} = x_{13} \oplus x_{15}, \\ x_{10} = x_6 \oplus x_8, & x_{14} = x_7 \oplus x_{12}, & x_{17} = x_{14} \wedge x_{16}. \end{array}$$

(f) This solution (found in 7471 Gμ) also matches that lower bound:

$$\begin{array}{lll} x_7 = x_1 \wedge x_2, & x_{11} = x_5 \oplus x_6, & x_{15} = x_8 \oplus x_{13}, \\ x_8 = x_1 \oplus x_2, & x_{12} = x_4 \oplus x_{11}, & x_{16} = x_{10} \oplus x_{14}, \\ x_9 = x_3 \oplus x_4, & x_{13} = x_9 \oplus x_{11}, & x_{17} = x_7 \oplus x_{16}, \\ x_{10} = x_5 \wedge x_6, & x_{14} = x_9 \vee x_{12}, & x_{18} = x_{15} \vee x_{17}. \end{array}$$

Here x_{18} is the normal function $\overline{S}_{0,4} = S_{1,2,3,5,6}$. We beat exercise 7.1.2–28 by one step.

(g) A solution in $t(3) = 12$ steps is found almost instantaneously (120 megamems); but 11 steps are too few ('unsat' in 301 gigamems).

480. (a) Let $x_1x_2x_3x_4 = x_lx_ry_ly_r$. The truth tables for z_l and z_r are 0011010010001000 and 01∗∗1∗00∗011∗011, where the ∗s ("don't-cares") are handled by simply *omitting* the corresponding clauses $(\bar{g}_{hi} \vee \pm x_{it})$ in answer 477.

Less than 1 gigamem of computation proves that a six-step circuit is 'unsat'. Here's a seven-stepper, found in just 30 Mμ: $x_5 = x_2 \oplus x_3$, $x_6 = x_3 \vee x_4$, $x_8 = x_1 \oplus x_6$, $x_7 = x_1 \vee x_5$, $x_9 = x_6 \oplus x_7$, $z_l = x_{10} = x_7 \wedge x_8$, $z_r = x_{11} = x_3 \oplus x_9$. (See exercise 7.1.2–60 for a six-step solution that is based on a *different* encoding.)

(b) Now we have the truth tables z_l = 00110100010010000100100010000011, z_r = 01∗∗1∗001∗00∗0111∗00∗011∗01101∗∗, if $x_4x_5 = y_ly_r$. One of many 9-step solutions is found in 6.9 gigamems: $x_6 = x_1 \oplus x_2$, $x_7 = x_2 \oplus x_5$, $x_8 = x_4 \oplus x_6$, $x_9 = \bar{x}_4 \wedge x_7$, $x_{10} = x_1 \oplus x_9$, $x_{11} = x_8 \vee x_9$, $x_{12} = x_3 \oplus x_{10}$, $z_r = x_{13} = x_3 \oplus x_{11}$, $z_l = x_{14} = x_{11} \wedge \bar{x}_{12}$.

The corresponding clauses for only 8 steps are proved 'unsat' after 190 Gμ of work. (Incidentally, the encoding of exercise 7.1.2–60 does *not* have a 9-step solution.)

(c) Let c_n be the minimum cost of computing the representation z_lz_r of $(x_1 + \cdots + x_n) \bmod 3$. Then $(c_1, c_2, c_3, c_4) = (0, 2, 5, 7)$, and $c_{n-3} \le c_n + 9$. Hence $c_n \le 3n - 4$ for all $n \ge 2$. [This result is due to A. Kojevnikov, A. S. Kulikov, and G. Yaroslavtsev, whose paper in *LNCS* **5584** (2009), 32–44, also inspired exercises 477–480.]

Conjecture: For $n \ge 3$ and $0 \le a \le 2$, the minimum cost of evaluating the (single) function $[(x_1 + \cdots + x_n) \bmod 3 = a]$ is $3n - 5 - [(n + a) \bmod 3 = 0]$. (It's true for $n \le 5$. Here's a 12-step computation when $n = 6$ and $a = 0$, found in 2014 by Armin Biere: $x_7 = x_1 \oplus x_2$, $x_8 = x_3 \oplus x_4$, $x_9 = x_1 \oplus x_5$, $x_{10} = x_3 \oplus x_5$, $x_{11} = x_2 \oplus x_6$, $x_{12} = x_8 \oplus x_9$, $x_{13} = x_8 \vee x_{10}$, $x_{14} = x_7 \oplus x_{13}$, $x_{15} = \bar{x}_{12} \wedge x_{13}$, $x_{16} = \bar{x}_{11} \wedge x_{14}$, $x_{17} = x_{11} \oplus x_{15}$, $\overline{S}_{0,3,6} = x_{18} = x_{16} \vee x_{17}$. The case $n = 6$ and $a \ne 0$, which lies tantalizingly close to the limits of today's solvers, is still unknown. What is $C(S_{1,4}(x_1, \ldots, x_6))$?)

481. (a) Since $z \oplus z' = \langle x_1x_2x_3 \rangle$ and $z' = x_1 \oplus x_2 \oplus x_3$, this circuit is called a "modified full adder." It costs one less than a normal full adder, since $z' = (x_1 \oplus x_2) \oplus x_3$ and $z = (x_1 \oplus x_2) \vee (x_1 \oplus x_3)$. (And it's the special case $u = 0$ of the more general situation in exercise 7.1.2–28.) Part (b) describes a "modified double full adder."

(b) The function z_2 has 20 don't-cares, so there are many eight-step solutions (although 7 are impossible); for example, $x_6 = x_1 \oplus x_5$, $x_7 = x_2 \oplus x_5$, $z_3 = x_8 = x_3 \oplus x_6$, $x_9 = x_4 \oplus x_6$, $x_{10} = x_1 \vee x_7$, $x_{11} = \bar{x}_3 \wedge x_9$, $z_2 = x_{12} = x_6 \oplus x_{11}$, $z_1 = x_{13} = x_{10} \oplus x_{11}$.

(c) Letting $y_{2k-1}y_{2k} = [\![x_{2k-1}x_{2k}]\!]$, it suffices to show that the binary representation of $\Sigma_n = \nu[\![y_1y_2]\!] + \cdots + \nu[\![y_{2n-1}y_{2n}]\!] + y_{2n+1}$ can be computed in at most $8n$ steps. Four steps are enough when $n = 1$. Otherwise, letting $c_0 = y_{2n+1}$, we can compute z's bits with $\nu[\![y_{4k-3}y_{4k-2}]\!] + \nu[\![y_{4k-1}y_{4k}]\!] + c_{k-1} = 2\nu[\![z_{2k-1}z_{2k}]\!] + c_k$ for $1 \le k \le \lfloor n/2 \rfloor$. Then $\Sigma_n = 2(\nu[\![z_1z_2]\!] + \cdots + \nu[\![z_{n-1}z_n]\!]) + c_{n/2}$ if n is even, $\Sigma_n = 2(\nu[\![z_1z_2]\!] + \cdots + \nu[\![z_{n-2}z_{n-1}]\!] + z_n) + c'$ if n is odd, where $\nu[\![y_{2n-1}y_{2n}]\!] + c_{\lfloor n/2 \rfloor} = 2z_n + c'$, at a cost of $4n$ in both cases. The remaining sum costs at most $8\lfloor n/2 \rfloor$ by induction. [See E. Demenkov, A. Kojevnikov, A. S. Kulikov, and G. Yaroslavtsev, *Information Processing Letters* **110** (2010), 264–267.]

482. (a) $\sum_{j=1}^{k}(2y_j - 1)$ is odd when k is odd, and it's ± 1 when $k = 1$.

(b) Adapting Sinz's cardinality clauses as in exercises 29 and 30, we only need the auxiliary variables $a_j = s_j^{j-1}$, $b_j = s_j^j$, and $c_j = s_j^{j+1}$, because $s_j^{j+2} = 0$ and $s_{j+2}^j = 1$. The clauses are then $(\bar{b}_j \vee a_{j+1}) \wedge (\bar{c}_j \vee b_{j+1}) \wedge (b_j \vee \bar{c}_j) \wedge (a_{j+1} \vee \bar{b}_{j+1})$, for $1 \le j < t/2 - 1$; and $(\bar{y}_{2j-2} \vee a_j) \wedge (\bar{y}_{2j-1} \vee \bar{a}_j \vee b_j) \wedge (\bar{y}_{2j} \vee \bar{b}_j \vee c_j) \wedge (\bar{y}_{2j+1} \vee \bar{c}_j) \wedge (y_{2j-2} \vee \bar{c}_{j-1}) \wedge (y_{2j-1} \vee c_{j-1} \vee \bar{b}_j) \wedge (y_{2j} \vee b_j \vee \bar{a}_{j+1}) \wedge (y_{2j+1} \vee a_{j+1})$ for $1 \le j < t/2$, omitting $\bar{a}_1$, c_0, and the two clauses that contain y_0.

don't-cares
Kojevnikov
Kulikov
Yaroslavtsev
Biere
modified full adder
full adder, modified
don't-cares
Demenkov
Kojevnikov
Kulikov
Yaroslavtsev
Sinz

(c) Use the construction in (b) with $y_j = x_{jd}$ for $1 \le d \le n/3$ and independent auxiliary variables $a_{j,d}$, $b_{j,d}$, $c_{j,d}$. Also, assuming that $n \ge 720$, break symmetry by asserting the unit clause (x_{720}). (That's much better than simply asserting (x_1).)

This problem was shown to be satisfiable if and only if $n < 1161$ by B. Konev and A. Lisitsa [*Artificial Intelligence* **224** (2015), 103–118], thereby establishing the case $C = 2$ of a well-known conjecture by Paul Erdős [*Michigan Math. J.* **4** (1957), 291–300, Problem 9]. Algorithm C can prove unsatisfiability for $n = 1161$ in less than 600 gigamems, using the parameters of exercise 512.

483. Using a direct encoding as in (15), with v_{jk} meaning that v_j has color k, we can generate the clauses $(\bar v_{jk})$ for $1 \le j < k \le d$ and $(\bar v_{j(k+1)} \vee \bigvee_{i=k}^{j-1} v_{ik})$ for $2 \le k < j \le n$. A similar but slightly simpler scheme works with the order encoding, when v_{jk} means that v_j has color $> k$. [See Ramani, Markov, Sakallah, and Aloul, *Journal of Artificial Intelligence Research* **26** (2006), 289–322. The vertices might be ordered in such a way that $\mathrm{degree}(v_1) \ge \cdots \ge \mathrm{degree}(v_n)$, for example.]

Those book graphs can be colored optimally with (11, 11, 13, 11, 10) colors, respectively. Such colorings are found with less than a megamem of work by Algorithm W or Algorithm C, *without* any symmetry breaking; Algorithm L also finds them, but after more than an order of magnitude more effort. The symmetry breaking clauses actually will *retard* this search, especially in the case of `homer`. On the other hand when we ask for only (10, 10, 12, 10, 9) colors those clauses are extremely helpful: The runtime for `anna` and `david` decreases from about 350 Gμ to only about 200 Kμ with Algorithm C! For `huck` and `jean` the reduction is roughly 333 Gμ $\to$ 833 Mμ and 14 Gμ $\to$ 4.3 Mμ; for `homer`, dozens or more of Tμ go down to about 11 Gμ. (Algorithm L is hopelessly slow on these unsatisfiable coloring problems, even with symmetry broken.)

484. (a) A type (iii) move will work if and only if $v_1 \,—\, v_4$, $v_2 \,—\, v_4$, $v_2 \,—\, v_3$.

(b) For $0 \le t < n-1$ we have the clause $(\bigvee_{k=1}^{n-t-1} q_{t,k} \vee \bigvee_{l=1}^{n-t-3} s_{t,l})$, as well as the following for $1 \le i < j < n-t$, $1 \le k < n-t$, $1 \le l < n-t-2$: $(\bar q_{t,k} \vee x_{t,k,k+1})$; $(\bar q_{t,k} \vee \bar x_{t+1,i,j} \vee x_{t,i',j'})$; $(\bar s_{t,l} \vee x_{t,l,l+3})$; $(\bar s_{t,k} \vee \bar x_{t+1,i,j} \vee x_{t,i'',j''})$; here $i' = i + [i \ge k]$, $j' = j + [j \ge k]$, and $\{i'', j''\}$ are the min and max of $\{i + [i \ge l+3] + 3[i = l], j + [j \ge l+3] + 3[j = l]\}$. Finally there's a unit clause $(\bar x_{0,i,j})$ for all $1 \le i < j \le n$ with $v_i \not\!\!—\, v_j$.

(These clauses essentially compute [G is quenchable], which is a monotone Boolean function of the $\binom{n}{2}$ elements above the diagonal in the adjacency matrix of G. The prime implicants of this function correspond to certain spanning trees, of which there are respectively 1, 1, 2, 6, 28, 164, 1137, ... when $n = 1, 2, 3, 4, 5, 6, 7, \ldots$.)

485. Let $t' = t+1$. Instances of commutativity are: $(q_{t,k}, q_{t',k'}) \leftrightarrow (q_{t,k'+1}, q_{t',k})$ if $k < k'$; $(s_{t,l}, s_{t',l'}) \leftrightarrow (s_{t,l'+1}, s_{t',l})$ if $l+2 < l'$; $(q_{t,k}, s_{t',l'}) \leftrightarrow (s_{t,l'+1}, q_{t',l})$ if $k < l'$; $(s_{t,l}, q_{t',k'}) \leftrightarrow (q_{t,k'+1}, s_{t',l})$ if $l+2 < k'$; $(s_{t,l}, s_{t',l}) \leftrightarrow (q_{t,l+3}, s_{t',l})$. These can be broken by appending the clauses $(\bar q_{t,k'+1} \vee \bar q_{t',k})$, $(\bar s_{t,l'+1} \vee \bar s_{t',l})$, ..., $(\bar q_{t,l+3} \vee \bar s_{t',l})$.

Endomorphisms are also present in the two cases $(q_{t,k}, q_{t',k}) \leftrightarrow (q_{t,k+1}, q_{t',k})$ and $(s_{t,k+1}, q_{t',k}) \leftrightarrow (q_{t,k+1}, s_{t',k})$, *provided* that *both* pairs of transitions are legal. These are exploited by the clauses $(\bar q_{t,k+1} \vee \bar q_{t',k} \vee \bar x_{t,k,k+1})$ and $(\bar q_{t,k+1} \vee \bar s_{t',k} \vee \bar x_{t,k+1,k+4})$.

486. This game is a special case of graph quenching, so we can use the previous two exercises. Algorithm C finds a solution after about 1.2 gigamems, without the symmetry-breaking clauses; this time goes down to roughly 85 *mega*mems when those clauses are added. Similarly, the corresponding 17-card problem after `A`♣ × `J`♣ is found to be unsatisfiable, after 15 Gμ without and 400 Mμ with. (`A`♣ ×× `10`♣ fails too.)

Those SAT problems have respectively (1242, 20392, 60905), (1242, 22614, 65590), (1057, 15994, 47740), (1057, 17804, 51571) combinations of (variables, clauses, cells),

auxiliary variables
break symmetry
Konev
Lisitsa
Erdős
direct encoding
order encoding
Ramani
Markov
Sakallah
Aloul
WalkSAT
comparison of running times
monotone Boolean function
adjacency matrix
prime implicants
Endomorphisms
quenching

and they are *not* handled easily by Algorithms A, B, D, or L. In one solution *both* $q_{0,11}$ and $s_{0,7}$ are true, thus providing two ways to win(!), when followed by $q_{1,15}$, $s_{2,13}$, $q_{3,12}$, $s_{4,10}$, $s_{5,7}$, $q_{6,7}$, $s_{7,5}$, $q_{8,5}$, $s_{9,4}$, $q_{10,5}$, $s_{11,3}$, $q_{12,3}$, $s_{13,1}$, $s_{14,1}$, $q_{15,1}$, $q_{16,1}$.

Notes: This mildly addictive game is an interesting way to waste time in case you ever get lost with a pack of cards on a desert island. If you succeed in reducing the original 18 piles to a single pile, you can continue by dealing 17 more cards and trying to reduce the new 18 piles. And if you succeed also at that, you have 17 more cards for a third try, since $52 = 18 + 17 + 17$. Three consecutive wins is a Grand Slam.

In a study of ten thousand random deals, just 4432 turned out to be winnable. Computer times (with symmetry breaking) varied wildly, from 1014 Kμ to 37 Gμ in the satisfiable cases (median 220 Mμ) and from 46 Kμ to 36 Gμ in the others (median 848 Mμ). The most difficult winnable and unwinnable deals in this set were respectively

9♠ 7♣ 3♣ K♢ 7♠ 3♡ 2♢ 8♣ 6♡ J♢ 8♠ 2♡ 6♠ 4♢ 5♠ 4♡ 10♢ Q♠ and
A♡ Q♡ 2♢ 9♢ 7♣ 7♢ 8♡ K♣ 3♢ 10♣ 3♣ 3♠ Q♠ 8♣ 2♣ K♠ 6♢ 5♣.

Students in Stanford's graduate problem seminar investigated this game in 1989 [see K. A. Ross and D. E. Knuth, Report STAN-CS-89-1269 (Stanford Univ., 1989), Problem 1]. Ross posed an interesting question, still unsolved: Is there a sequence of (say) nine "poison cards," such that all games starting with those cards are lost?

The classic game Idle Year is also known by many other names, including Tower of Babel, Tower of London, Accordion, Methuselah, and Skip Two. Albert H. Morehead and Geoffrey Mott-Smith, in *The Complete Book of Solitaire and Patience Games* (1949), 61, suggested that moves shouldn't be too greedy.

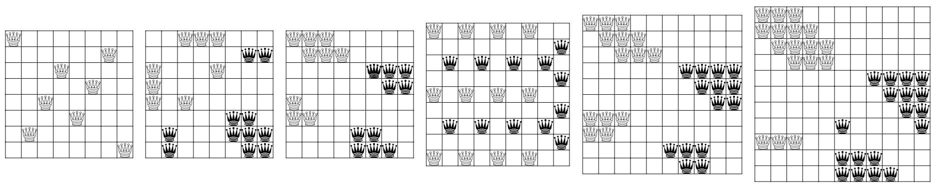
487. Every queen in a set of eight must attack at least 14 vacant cells. Thus $|\partial S|$ gets its minimum value $8 \times 14 = 112$ when the queens occupy the top row. Solutions to the 8 queens problem, when queens are independent, all have $|\partial S| \le 176$. The maximum $|\partial S|$ is 184, achieved symmetrically for example in Fig. A–9(a). (This problem is *not* at all suitable for SAT solvers, because the graph has 728 edges. The best way to proceed is to run through all $\binom{64}{8}$ possibilities with the revolving door Gray code (Algorithm 7.2.1.3R), because incremental changes to $|\partial S|$ are easy to compute when a queen is deleted or inserted. The total time by that method is only 601 gigamems.)

The maximum of $|\partial_{\rm out} S|$ is obviously $64 - 8 = 56$. The minimum, which corresponds to Turton's question, is 45; it can be achieved symmetrically as in Fig. A–9(b), leaving $64 - 8 - 45 = 11$ cells unattacked (shown as black queens). In this case SAT solvers win: The revolving door method needs 953 gigamems, but SAT methods show the impossibility of 44 after only 2.2 Gμ of work. With symmetry reduction as in the following exercise, this goes down to 900 Mμ although there are 789 variables and 4234 clauses. [Bernd Schwarzkopf, in *Die Schwalbe* **76** (August 1982), 531, computed all solutions of minimum $|\partial_{\rm out} S|$, given $|S|$, for $n \times n$ boards with $n \le 8$. Extensions of Turton's problem to larger n have been surveyed by B. Lemaire and P. Vitushinskiy in two articles, written in 2011 and accessible from `www.ffjm.org`. Optimum solutions for $n > 16$ are conjectured but not yet known.]

All sets S of eight queens trivially have $|\partial_{\rm in} S| = 8$.

488. Let variables w_{ij} and b_{ij} represent the presence of white or black queens on cell (i,j), with clauses $(\bar w_{ij} \vee \bar b_{i'j'})$ when $(i,j) = (i',j')$ or $(i,j) \mathrel{—} (i',j')$. Also, if each army is to have at least r queens, add clauses based on (20) and (21) to ensure that $\sum w_{ij} \ge r$ and $\sum b_{ij} \ge r$. Optionally, add clauses based on Theorem E to ensure that k of the w variables for the top row are lexicographically greater than or equal to the

Stanford
Ross
Knuth
poison cards
Tower of Babel
Tower of London
Accordion
Methuselah
Skip Two
Morehead
Mott-Smith
8 queens problem
revolving door
Gray code
symmetry reduction
Schwarzkopf
Lemaire
Vitushinskiy
lexicographically



(a) (b) (c) (d) (e) (f)

Fig. A–9. Optimum queen placements of various kinds.

corresponding k variables in fifteen symmetrical variants. (For instance, if $k = 3$, we might require $w_{11}w_{12}w_{13} \geq b_{1n}b_{2n}b_{3n}$, thus partially breaking the symmetries.)

The maximum army sizes for $1 \leq n \leq 11$ are found to be 0, 0, 1, 2, 4, 5, 7, 9, 12, 14, and 17, respectively. A construction with 21 armies is known for $n = 12$, but 22 has not yet been proved impossible. [B. M. Smith, K. E. Petrie, and I. P. Gent obtained similar results using CSP methods in *LNCS* **3011** (2004), 271–286.] An extra black queen can actually be included in the cases $n = 2$, 3, 4, 6, 8, 10, and 11. Solutions appear in Fig. A–9; the construction shown in Fig. A–9(d) generalizes to armies of $2q(q + 1)$ queens whenever $n = 4q + 1$, while those in parts (c), (e), (f) belong to another family of constructions that achieve the higher asymptotic density $\frac{9}{64}n^2$.

When $n = 8$ and $r = 9$, Algorithm C typically finds a solution in about 10 megamems ($k = 0$), or about 30 megamems ($k = 3$); but with $r = 10$ it typically proves unsatisfiability in about 1800 Mμ ($k = 0$) or 850 Mμ ($k = 3$) or 550 Mμ ($k = 4$) or 600 Mμ ($k = 5$). Thus the symmetry breaking constraints are helpful for unsatisfiability in this case, but not for the easier satisfiability problem. On the other hand, the extra constraints do turn out to be helpful for both the satisfiable and unsatisfiable variants when n is larger. The "sweet spot" turns out to be $k = 6$ when $n = 10$ and $n = 11$; unsatisfiability was proved in those cases, with $r = 15$ and $r = 18$, after about 185 Gμ and 3500 Gμ, respectively. [See Martin Gardner, *Math Horizons* **7**, 2 (November 1999), 2–16, for generalizations to coexisting armies of sizes r and s. F. R. K. Chung and R. L. Graham conjecture that the maximum value of s, if $r = 3q^2 + 3q + 1$, is asymptotically $n^2 - (6q + 3)n + O(1)$.]

489. $T_0 = 1$, $T_1 = 2$, $T_n = 2T_{n-1} + (2n - 2)T_{n-2}$ (see Eq. 5.1.4–(40)). The generating function $\sum_n T_n z^n/n!$ and the asymptotic value are given in exercise 5.1.4–31.

490. Yes. For example, using the signed permutation $\bar{4}13\bar{2}$, we're allowed to assume that some solution satisfies $\bar{x}_4x_1x_3\bar{x}_2 \leq \bar{x}'_4x'_1x'_3\bar{x}'_2$ for every endomorphism — because the solution with lexicographically smallest $\bar{x}_4x_1x_3\bar{x}_2$ has this property. Notice that the signed permutation $\bar{1}\bar{2}\ldots\bar{n}$ converts '$\leq$' to '$\geq$'.

491. Let σ be the permutation $(1\,2\,3\,4\,\bar{1}\,\bar{2}\,\bar{3}\,\bar{4})$. Then $\sigma^4 = (1\,\bar{1})(2\,\bar{2})(3\,\bar{3})(4\,\bar{4})$; and by Theorem E we need only search for solutions that satisfy $x_1x_2x_3x_4 \leq \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4$. We're therefore allowed to append the clause $(\bar{x}_1)$ without affecting satisfiability.

(We actually are allowed to assert that $x_1 = x_2 = x_4 = 0$, because 0000 and 0010 are the lex-leaders of the two 8-cycles when σ is a written as a permutation of states.)

In general if an automorphism σ is a permutation of literals having a cycle that contains both v and $\bar{v}$, for some variable v, we can simplify the problem by assigning a fixed value to v and then by restricting consideration to automorphisms that don't change v. (See the discussion of Sims tables in Section 7.2.1.2.)

492. Suppose $x_1 \ldots x_n$ satisfies all clauses of F; we want to prove that $(x_1 \ldots x_n)\tau = x'_1 \ldots x'_n$ also satisfies them all. And that's easy: If $(l_1 \vee \cdots \vee l_k)$ is a clause, we have

breaking the symmetries
Smith
Petrie
Gent
CSP: The constraint satisfaction problem
asymptotic
Gardner
Chung
Graham
lex-leaders
Sims tables

$l'_1 = l_1\tau, \ldots, l'_n = l_n\tau$; and we know that $(l_1\tau \vee \cdots \vee l_k\tau)$ is true because it's subsumed by a clause of F. [See S. Szeider, *Discrete Applied Math.* **130** (2003), 351–365.]

493. Using the global ordering $p_1 \ldots p_9 = 543219876$ and Corollary E, we can add clauses to assert that $x_5 = 0$ and $x_4x_3x_2x_1 \le x_6x_7x_8x_9$. A contradiction quickly follows, even if we stipulate only the weaker relation $x_4 \le x_6$, because that forces $x_6 = 1$.

494. Exercise 504(b) shows that $(uv)(\bar{u}\bar{v})$ is a symmetry of the underlying Boolean function, although not necessarily of the clauses F. [This observation is due to Aloul, Ramani, Markov, and Sakallah in the cited paper.] The other symmetries allow us to assert (i) $(\bar{x}_i \vee x_j) \wedge (\bar{x}_j \vee \bar{x}_k)$, (ii) $(\bar{x}_i \vee \bar{x}_j) \wedge (\bar{x}_j \vee \bar{x}_k)$, (iii) $(\bar{x}_i \vee \bar{x}_j) \wedge (\bar{x}_j \vee x_k)$.

495. Suppose, for example, that $m = 3$ and $n = 4$. The variables can then be called 11, 12, 13, 14, 21, ..., 34; and we can give them the global ordering 11, 12, 21, 13, 22, 31, 14, 23, 32, 24, 33, 34. To assert that $21\,22\,23\,24 \le 31\,32\,33\,34$, we use the involution that swaps rows 2 and 3; this involution is $(21\,31)(22\,32)(23\,33)(24\,34)$ when expressed in form (192) with signs suppressed. Similarly we can assert that $12\,22\,13 \le 13\,23\,33$ because of the involution $(12\,13)(22\,23)(32\,33)$ that swaps columns 2 and 3. The same argument works for any adjacent rows or columns. And we can replace '$\le$' by '$\ge$', by complementing all variables.

For general m and n, consider any global ordering for which x_{ij} precedes or equals $x_{i'j'}$ when $1 \le i \le i' \le m$ and $1 \le j \le j' \le n$. The operation of swapping adjacent rows makes the global lexicographic order increase if and only if it makes the upper row increase lexicographically; and the same holds for columns.

[See Ilya Shlyakhter, *Discrete Applied Mathematics* **155** (2007), 1539–1548.]

496. No; that reasoning would "prove" that m pigeons cannot fit into m holes. The fallacy is that his orderings on rows and columns aren't simultaneously consistent with a *single* global ordering, as in the previous exercise.

497. A BDD with 71,719 nodes makes it easy to calculate the total, 818,230,288,201, as well as the generating function $1 + z + 3z^2 + 8z^3 + 25z^4 + \cdots + 21472125415z^{24} + 31108610146z^{25} + \cdots + 10268721131z^{39} + 6152836518z^{40} + \cdots + 24z^{60} + 8z^{61} + 3z^{62} + z^{63} + z^{64}$. (The relatively small coefficients of z^{39} and z^{40} help account for the fact that $\ge$ was chosen in (185)–(186); problems with sparse solutions tend to favor $\ge$.)

[Pólya's theorem in Section 7.2.3 shows that exactly 14,685,630,688 inequivalent matrices exist; compare this to $2^{64} \approx 1.8447 \times 10^{19}$ without any symmetry reduction.]

498. Consider the global ordering $x_{01}, x_{11}, \ldots, x_{m1}$; $x_{12}, x_{22}, \ldots, x_{m2}, x_{02}$; $x_{23}, x_{33}, \ldots, x_{m3}, x_{03}, x_{13}$; ...; $x_{(m-1)m}, x_{mm}, x_{0m}, \ldots, x_{(m-2)m}$. There's a column symmetry that fixes all elements preceding $x_{(j-1)j}$ and takes $x_{(j-1)j} \mapsto x_{(j-1)k}$.

499. No. The unusual global ordering in answer 498 is not consistent with ordinary lexicographic row or column ordering. [Nor can the analogous clauses $(x_{ii} \vee \bar{x}_{ij})$ for $1 \le i \le m$ and $i < j \le n$ be appended to (185) and (186). No quad-free matrix for $m = n = 4$ and $r = 9$ satisfies all those constraints simultaneously.]

500. If F_0 has a solution, then it has a solution for which l is true. But $(F_0 \cup F_1) \mid l$ might be unsolvable. (For example, let $F_0 = (\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee x_1)$, which has the symmetry $\bar{1}\bar{2}$; so we can take $S = (\bar{x}_1)$, $l = \bar{x}_1$. Combine that with $F_1 = (x_1)$.)

501. Let x_{ij} denote a queen in cell (i, j), for $1 \le i \le m$ and $1 \le j \le n$. Also let $r_{ij} = [x_{i1} + \cdots + x_{ij} \ge 1]$ and $r'_{ij} = [x_{i1} + \cdots + x_{i(j+1)} \ge 2]$, for $1 \le i \le m$ and $1 \le j < n$. Using (18) and (19) we can easily construct about $8mn$ clauses that define the r's in terms of the x's and also ensure that $x_{i1} + \cdots + x_{in} \le 2$. Thus $r'_{i(n-1)} = [x_{i1} + \cdots + x_{in} = 2]$; call this condition r_i.

Szeider
Aloul
Ramani
Markov
Sakallah
Shlyakhter
fallacy
generating function
Pólya's theorem
global ordering
quad-free

Similar conditions c_j, a_d, and b_d are readily established for column j, and for the diagonals with $i+j = d+1$ or $i-j = d-n$, for $1 \le i \le m$, $1 \le j \le n$, and $1 \le d < m+n$. Then condition (ii) corresponds to the mn clauses $(x_{ij} \vee r_i \vee c_j \vee a_{i+j-1} \vee b_{i-j+n})$.

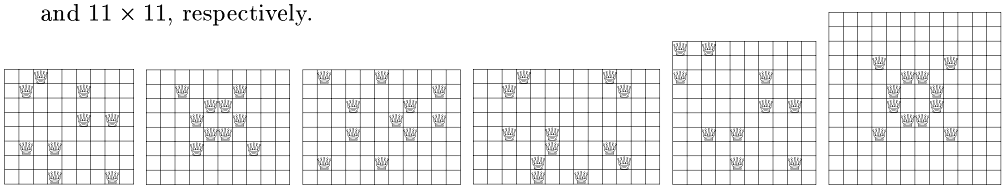
Finally we have clauses from (20) and (21) to ensure that $\sum x_{ij} \le r$.

When $m = n$, the lower bound $r \ge n - [n \bmod 4 = 3]$ has been established by A. S. Cooper, O. Pikhurko, J. R. Schmitt, and G. S. Warrington [*AMM* **121** (2014), 213–221], who also used backtracking to show that $r \ge 12$ on an 11×11 board. SAT methods, with symmetry breaking, yield that result much more quickly (after about 9 teramems of computation); but this problem, like the tomography problem of Fig. 36, is best solved by integer programming techniques when m and n are large.

Cooper
Pikhurko
Schmitt
Warrington
symmetry breaking
tomography problem
integer programming
magic sequences
cardinality constraints
Hamming distance
Karpovsky
Frances
Litman

If we call the upper left corner white, solutions with $m = n = r-1$ and all queens on white squares appear to exist for all $n > 2$, and they are found almost instantly. However, no general pattern is apparent. In fact, when n is odd it appears possible to insist that the queens all appear in odd-numbered rows *and* in odd-numbered columns.

Here are examples of optimum placements on smallish boards. The solutions for 8×9, 8×10, 8×13, 10×10, and 12×12 also work for sizes 8×8, 9×10, 8×12, 9×9, and 11×11, respectively.



This placement of ten queens on a 10×10 board can be described by the "magic sequence" $(a_1, \ldots, a_5) = (1, 3, 7, 5, 9)$, because the queens appear in positions (a_i, a_{i+1}) and (a_{i+1}, a_i) for $1 \le i < n/2$ as well as in (a_1, a_1) and $(a_{n/2}, a_{n/2})$. The magic sequences (1, 3, 9, 13, 15, 5, 11, 7, 17) and (9, 3, 1, 19, 5, 11, 15, 25, 7, 21, 23, 13, 17) likewise describe optimum placements for $n = 18$ and 26. No other magic sequences are known; none exist when $n = 34$.

502. For each j, construct independent cardinality constraints for the relation $x_1^{(j)} + \cdots + x_n^{(j)} \le r_j$, using say (20) and (21), where $x_k^{(j)} = (s_{jk}?\ \bar{x}_k\colon x_k)$.

503. The Hamming distance $d(x, y) = \nu(x \oplus y)$ between binary vectors of length n satisfies $d(x, y) + d(\bar{x}, y) = n$. Thus there is no x with $d(x, s_j) \ge r_j + 1$ for all j if and only if there is no x with $d(\bar{x}, s_j) \le n - 1 - r_j$ for all j. [See M. Karpovsky, *IEEE Transactions* **IT-27** (1981), 462–472.]

504. (a) Assume that $n \ge 4$. For strings of length $2n$ we have $d(z, w) + d(z, \bar{w}) = 2n$; hence $d(z, w) \le n$ and $d(z, \bar{w}) \le n$ if and only if $d(z, w) = d(z, \bar{w}) = n$. Every string z with $z_{2k-1} \ne z_{2k}$ for $1 \le k \le n$ satisfies $d(z, w_j) = n$ for $1 \le j \le n$. Conversely, if $d(z, w_j) = d(z, w_k) = n$ and $1 \le j < k \le n$, then $z_{2j-1} + z_{2j} = z_{2k-1} + z_{2k}$. Thus if $z_{2j-1} = z_{2j}$ for some j we have $z = 00\ldots0$ or $11\ldots1$, contradicting $d(z, w_1) = n$.

(b) For each string $\hat{x} = \bar{x}_1 x_1 \bar{x}_2 x_2 \ldots \bar{x}_n x_n$ that satisfies part (a) we have $d(\hat{x}, y) = 2\bar{l}_1 + 2\bar{l}_2 + 2\bar{l}_3 + n - 3$, which is $\le n + 1$ if and only if $(l_1 \vee l_2 \vee l_3)$ is satisfied.

(c) Let $s_j = w_j$ and $r_j = n$ for $1 \le j \le 2n$; let $s_{2n+k} = y_k$ and $r_{2n+k} = n + 1$ for $1 \le k \le m$, where y_k is the string in (b) for the kth clause of F. This system has a closest string $\hat{x} = \bar{x}_1 x_1 \bar{x}_2 x_2 \ldots \bar{x}_n x_n$ if and only if $x_1 \ldots x_n$ satisfies every clause. [A similar construction in which all strings have length $2n + 1$ and all r_j are equal to $n + 1$ is obtained if we append the bit $[n < j \le 2n]$ to each s_j. See M. Frances and A. Litman, *Theory of Computing Systems* **30** (1997), 113–119.]

(d) Boilerplate 11000000, 00110000, 00001100, 00000011, 00111111, 11001111, 11110011, 00000011, at distance ≤ 4; for the clauses, 01011000, 00010110, 01000101, 10010001, 10100100, 00101001, 10001010, and possibly 01100010, at distance ≤ 5.

505. (For $k = 0, 1, \ldots, n-1$ one can set j to a uniform integer in $[0\,..\,k]$ and $\texttt{INX}[k+1] \leftarrow j$; also if $j = k$ set $\texttt{VAR}[k] \leftarrow k+1$, otherwise $i \leftarrow \texttt{VAR}[j]$, $\texttt{VAR}[k] \leftarrow i$, $\texttt{INX}[i] \leftarrow k$, $\texttt{VAR}[j] \leftarrow k+1$.) With nine random seeds, typical runtimes for D3 are (1241, 873, 206, 15, 748, 1641, 1079, 485, 3321) Mμ. They're much less variable for the unsatisfiable K0, namely (1327, 1349, 1334, 1330, 1349, 1322, 1336, 1330, 1317) Mμ; and even for the satisfiable W2: (172, 192, 171, 174, 194, 172, 172, 170, 171) Mμ.

506. (a) *Almost* true: That sum is the total number of clauses of length ≥ 2, because every such clause of length k contributes $1/\binom{k}{2}$ to the weights of $\binom{k}{2}$ edges.

(b) Each of the $12^2 - 2 = 142$ cells of the mutilated 12×12 board contributes one positive clause $(v_1 \vee \cdots \vee v_k)$ and $\binom{k}{2}$ negative clauses $(\bar{v}_i \vee \bar{v}_j)$, when that cell can be covered by k potential dominoes $\{v_1, \ldots, v_k\}$. So the weight between u and v is 2, 4/3, or 7/6 when dominoes u and v overlap in a cell that can be covered in 2, 3, or 4 ways. Exactly 6 cells can be covered in just 2 ways (and exactly 10^2 in 4 ways).

(The largest edge weights in all of Fig. 52 are 37/6, between 20 pairs of vertices in K6. At the other extreme, 95106 of the 213064 edges in X3 have the tiny weight 1/8646, and 200904 of them have weight at most twice that much.)

507. Consider, for example, the clauses $(u \vee \bar{t})$, $(v \vee \bar{t})$, $(\bar{u} \vee \bar{v} \vee t)$, $(u \vee \bar{t}')$, $(v \vee \bar{t}')$, $(\bar{u} \vee \bar{v} \vee t')$ from (24). Looking ahead from $t = 1$ yields the windfall $(\bar{t} \vee t')$, and looking ahead from $t' = 1$ yields $(\bar{t}' \vee t)$. Henceforth Algorithm L knows that t equals t'.

508. According to (194), the purging parameters were $\Delta_{\text{p}} = 1000$ and $\delta_{\text{p}} = 500$; thus we have learned approximately $1000k + 500\binom{k}{2}$ clauses when doing the kth purging phase. After $1000L$ clauses this works out to be $\approx (\sqrt{16L+9} - 3)/2$ phases, which is ≈ 34.5 when $L = 323$. (And the actual number was indeed 34.)

509. One remedy for overfitting is to select training examples at random. In this case such randomness is already inherent, because of the different seeds used while training.

510. (a) From Fig. 53 or Fig. 54 or Table 7 we know that T1 < T2 < L6 in the median rankings; thus T2 obscures L6 and T1.

(b) Similarly, L8 < M3 < Q2 < X6 < F2 < X4 < X5; X6 obscures L8 and X4.

(c) X7 obscures K0, K2, and (indirectly) A2, because K2 obscures K0 and A2.

511. (a) Nine random runs finished in only $(4.9, 5.0, 5.1, 5.1, 5.2, 5.2, 5.3, 5.4, 5.5)$ Mμ(!).

(b) Nine random runs now each were aborted after a teramem of trials. (No theoretical explanation for this discrepancy, or for the wildness of P4 in Fig. 54, is known.)

(c) $(0.2, \ldots, 0.5, \ldots, 3.2)$ Mμ without; $(0.3, \ldots, 0.5, \ldots, 0.7)$ Mμ with.

512. A training run with ParamILS in 2015 suggested the parameters

$$\alpha = 0.7,\ \ \rho = 0.998,\ \ \varrho = 0.99995,\ \ \Delta_{\text{p}} = 100000,\ \ \delta_{\text{p}} = 2000,$$
$$\tau = 10,\ \ w = 1,\ \ p = 0,\ \ P = 0.05,\ \ \psi = 0.166667, \qquad (*)$$

which produce the excellent results in Fig. A–10.

513. After training on *rand*$(3, 1062, 250, 314159)$, ParamILS choose the values $\alpha = 3.5$ and $\Theta = 20.0$ in (195), together with distinctly different values that favor double lookahead, namely $\beta = .9995$, $Y = 32$. [The untuned values $\alpha = 3.3$, $\beta = .9985$, $\Theta = 25.0$, and $Y = 8$ had been used by the author when preparing exercise 173.]

mutilated
Looking ahead
ParamILS
pi
double lookahead
author

ParamILS

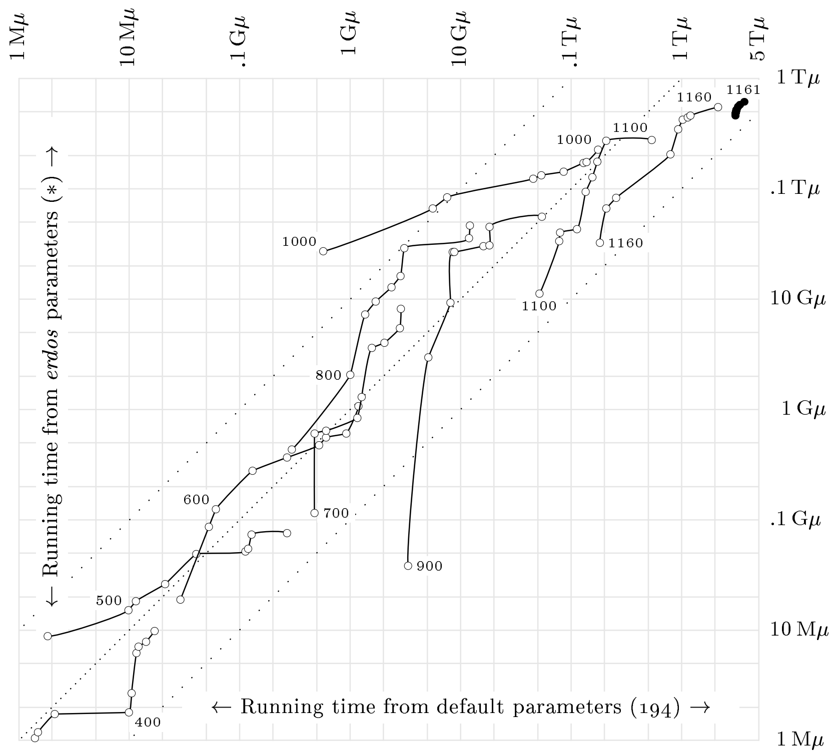


Fig. A–10. Running times for Algorithm C, with and without special parameter tuning.

514. ParamILS suggests $p = .85$ and $N = 5000n$; that gives a median time $\approx 690\,\mathrm{M}\mu$. (But those parameters give horrifically *bad* results on most other problems.)

515. Use variables S_{ijk} meaning that cell (i,j) in the solution holds k, and Z_{ij} meaning that cell (i,j) is blank in the puzzle. The 729 S variables are constrained by $4 \times 81 \times (1 + \binom{9}{2}) = 11{,}988$ clauses like (13). From condition (i), we need only 41 variables Z_{ij}. Condition (ii) calls for 15 clauses such as $(Z_{11} \vee \cdots \vee Z_{19})$, $(Z_{11} \vee \cdots \vee Z_{51} \vee Z_{49} \vee \cdots \vee Z_{19})$, $(Z_{15} \vee \cdots \vee Z_{55})$, $(Z_{44} \vee Z_{45} \vee Z_{46} \vee Z_{54} \vee Z_{55})$, when equal Z's are identified via (i). Condition (iii), similarly, calls for 28 clauses such as $(\bar{Z}_{11} \vee \bar{Z}_{12} \vee \bar{Z}_{13})$, $(\bar{Z}_{11} \vee \bar{Z}_{21} \vee \bar{Z}_{31})$, $(\bar{Z}_{45} \vee \bar{Z}_{55})$. Condition (vi) is enforced by 34,992 clauses epitomized by $(\bar{S}_{111} \vee \bar{Z}_{11} \vee \bar{S}_{122} \vee \bar{Z}_{12} \vee \bar{S}_{412} \vee \bar{Z}_{41} \vee \bar{S}_{421} \vee \bar{Z}_{42})$.

For conditions (iv) and (v), we introduce auxiliary variables $V_{ijk} = S_{ijk} \wedge \bar{Z}_{ij}$, meaning that k is visible in (i,j); $R_{ik} = V_{i1k} \vee \cdots \vee V_{i9k}$, meaning that k is visible in row i; $C_{jk} = V_{1jk} \vee \cdots \vee V_{9jk}$, meaning that k is visible in column j. Also $B_{bk} = \bigvee_{\langle i,j\rangle = b} V_{ijk}$, meaning that k is visible in box b; here $\langle i,j\rangle = 1 + 3\lfloor (i-1)/3 \rfloor + \lfloor (j-1)/3 \rfloor$. Then $P_{ijk} = Z_{ij} \wedge \bar{R}_{ik} \wedge \bar{C}_{jk} \wedge \bar{B}_{\langle i,j\rangle k}$ means that k is a possible way to fill cell (i,j) without conflict. These 1701 auxiliary variables are defined with 8262 clauses.

Condition (iv) is enforced by nine 9-ary clauses for each i and j, stating that we mustn't have exactly one of $\{P_{ij1}, \ldots, P_{ij9}\}$ true. Condition (v) is similar, enforced by three sets of 81×9 clauses of length 9; for example, one of those clauses is

$$(P_{417} \vee P_{427} \vee P_{437} \vee P_{517} \vee \bar{P}_{527} \vee P_{537} \vee P_{617} \vee P_{627} \vee P_{637}).$$

("We aren't obviously forced to put 7 into box 4 by using cell $(5, 2)$.")

Finally, some of the symmetry is usefully broken by asserting the unary clauses $S_{1kk} \wedge \bar{Z}_{11} \wedge Z_{12}$. The grand total is 58,212 clauses with 351,432 cells, on 2,471 variables.

(This problem was suggested by Daniel Kroening. There are zillions of solutions, and about one in every five or six appears to be completable uniquely to the setting of the S variables. Thus we can obtain as many "hard sudoku" puzzles as we like, by adding additional unary clauses such as $S_{553} \wedge \bar{Z}_{17}$ more or less at random, then weeding out ambiguous cases via dancing links. The clauses are readily handled by Algorithms L or C, but they're often too difficult for Algorithm D. That algorithm did, however, find the uniquely completable solution (a) below after only 9.3 gigamems of work.)

If we beef up condition (iii), insisting now that no box contains a row or column with more than *one* blank, condition (vi) becomes superfluous. We get solutions such as (b) below, remarkable for having no forced moves in spite of 58 visible clues, yet uniquely completable. That puzzle is, however, quite easy; only 2, 4, 7 are unplaced.

(a)	(b)	(c)	(d)
`1....6.8.`	`1.3.56.89`	`1.3.5.7..`	`1.3.56.89`
`5.87214.6`	`59738.61.`	`.5.79...1`	`68.3.91.5`
`.6.38.2.1`	`68.1.93.5`	`7....125.`	`.9518.63.`
`84...3..5`	`956.318.7`	`..1..5.76`	`3.896..51`
`..5.6.8..`	`.315.896.`	`..5.7.1..`	`.195.836.`
`6..8...42`	`2.896.153`	`47.1..5..`	`56..319.8`
`3.6.48.2.`	`8.96.5.31`	`.185....7`	`.56.9381.`
`4.76321.8`	`.65.13298`	`5...87.1.`	`8.16.5.93`
`.8.5....4`	`31.89.5.6`	`..7.1.8.5`	`93.81.5.6`

We might also try to strengthen conditions (iv) and (v) by requiring at least *three* ways to make each choice, not just two. Then we get solutions like (c) above. Unfortunately, however, that one is completable in 1237 ways! Even if we also strengthen condition (iii) as in (b), we get solutions like (d), which can be completed in 12 ways. No uniquely completable sudoku puzzles are known to have such ubiquitous threefold ambiguity.

516. This conjecture can be expressed in several equivalent forms. R. Impagliazzo and R. Paturi [*JCSS* **62** (2001), 367–375] defined $s_k = \inf\{\lg \tau \mid$ there exists an algorithm to solve kSAT in τ^n steps$\}$, and stated the *exponential time hypothesis*: $s_3 > 0$. They also defined $s_\infty = \lim_{k\to\infty} s_k$, and proved that $s_k \le (1 - d/k)s_\infty$ for some positive constant d. They conjectured that $s_\infty = 1$; this is the strong exponential time hypothesis. An alternative formulation [C. Calabro, R. Impagliazzo, and R. Paturi, *IEEE Conf. on Computational Complexity* **21** (2006), 252–260] was found later: "If $\tau < 2$, there is a constant α such that no randomized algorithm can solve every SAT problem with $\le \alpha n$ clauses in fewer than τ^n steps, where n is the number of variables."

517. (a) If there are n variables, introduce $\binom{2n}{2}$ new variables $ll' = l'l$, one for each pair of literals $\{l, l'\}$, with the equations $ll' + l\bar{l}' + \bar{l} = 1$. Similarly, introduce $\binom{2n}{3}$ variables $ll'l''$, via $ll'l'' + ll'\bar{l}'' + l\bar{l}' + \bar{l} = 1$. Then the ordinary ternary clause $l \vee l' \vee l''$ is true if and only if we have $ll'l'' + ll'\bar{l}'' + l\bar{l}'l'' + l\bar{l}'\bar{l}'' + \bar{l}l'l'' + \bar{l}l'\bar{l}'' + \bar{l}\bar{l}'l'' = 1$.

(b) Remove clauses of length > 3 by using the fact that $l_1 + \cdots + l_k = 1$ if and only if $l_1 + \cdots + l_j + t = 1$ and $l_{j+1} + \cdots + l_k + \bar{t} = 1$, where t is a new variable. Also, if a, b, c, and d are new variables with $a + b + d = a + c + \bar{d} = 1$, beef up short clauses using $l + l' = 1 \iff l + l' + a = 1$ and $l = 1 \iff \bar{l} + b + c = 1$.

symmetry breaking
Kroening
dancing links
Impagliazzo
Paturi
exponential time hypothesis
Calabro
density of clauses

[Thomas J. Schaefer proved the NP-completeness of 1-in-3 SAT as a special case of considerably more general results, in *STOC* **10** (1978), 216–226.]

518. (a) $A = \left(\begin{smallmatrix} x & y & y \\ y & x & y \\ y & y & x \end{smallmatrix}\right)$, where $x = \begin{smallmatrix} -1 & 0 \\ 1 & 0 \end{smallmatrix}$, $y = \begin{smallmatrix} 1 & 1 \\ -1 & 1 \end{smallmatrix}$.

(b) Twice in the n variable rows and n variable columns; once in the $3m$ output rows and $3m$ input columns; never in the $3m$ input rows and $3m$ output columns.

(c) By (a), each way to choose 2s in different rows and columns contributes zero to the permanent unless, in every clause, the subset of chosen inputs is nonempty and matches the chosen outputs. In the latter case it contributes $16^m 2^n$. [See A. Ben-Dor and S. Halevi, *Israel Symp. Theory of Computing Systems* **2** (IEEE, 1993), 108–117.]

519. The unsatisfiable problem corresponding to D1 and D2 has median running time 2099 Mμ (losing to both *factor_fifo* and *factor_lifo*). The satisfiable one corresponding to D3 and D4 is unstable (as in Fig. 54), with median 903 Mμ (winning over both).

520. (Solution by Sven Mallach, 2015, using solvers X and Y, where X was CPLEX 12.6 and Y was GUROBI 6, both used with emphasis on mixed-integer-program feasibility, constant objective function, and solution limit 1.) With a time cutoff of 30 minutes on a single-threaded Xeon computer, neither X nor Y could solve any of the 46 problems A1, A2, C1, C2, C3, C4, C5, C6, C8, D1, D2, E1, E2, F1, F2, G1, G2, G5, G6, G7, G8, K7, K8, M5, M7, M8, O1, O2, P0, P1, P2, Q7, S3, S4, T5, T6, T7, T8, W2, W4, X1, X3, X5, X6, X7, X8. (In particular, this list includes P0, S4, and X1, which are extremely easy for Algorithm C.) On the other hand both X and Y solved the *langford* problems L3 and L4 — which were the toughest for Algorithm C — in less than a second.

Algorithm C performs about 20 Gμ per minute on a comparable Xeon. In these experiments it significantly outperformed the geometric methods except on problems K0, K1, K2, L3, L4, and P4 (and some easy cases such as B2).

Of course we must keep in mind that the particular clauses in Table 6 aren't necessarily the best ways to solve the corresponding combinatorial problems with an IP solver, just as they aren't necessarily the best encodings for a SAT solver. We are comparing here only black-box clause-solving speeds.

521. A variety of simple schemes has been surveyed by S. Jabbour, J. Lonlac, L. Saïs, and Y. Salhi, `arXiv:1402.1956` [cs.AI] (2014), 13 pages.

522. For cycles of length T we can introduce $27T$ variables xyz_t for $1 \le x, y, z \le 3$ and $0 \le t < T$, signifying that vertex (x, y, z) occupies slot t in the path. Binary exclusion clauses $\neg xyz_t \vee \neg x'y'z'_{t'}$, when $xyz = x'y'z'$ and $t \ne t'$ or when $xyz \ne x'y'z'$ and $t = t'$, ensure that no vertex appears twice in the path, and that no two vertices occupy the same slot. A valid path is specified via the adjacency clauses

$$\neg xyz_t \;\vee\; \bigvee\{x'y'z'_{(t+1) \bmod T} \mid 1 \le x', y', z' \le 3 \text{ and } |x'-x| + |y'-y| + |z'-z| = 1\}.$$

We represent the shadows by introducing 36 variables $a!b{*}$, $ba!{*}$, $a!{*}b$, $b{*}a!$, ${*}a!b$, ${*}ba!$ for $1 \le a \le 2$ and $1 \le b \le 3$; here $a!{*}b$ (for example) means that the shadow of (x, z) coordinates has a transition between (a, b) and $(a{+}1, b)$. These variables appear in ternary clauses such as $(\neg xyz_t \vee \neg(x{+}1)yz_{t'} \vee x!{*}z) \wedge (\neg xyz_t \vee \neg(x{+}1)yz_{t'} \vee x!y{*})$ whenever $x < 3$ and $t' \equiv t \pm 1$ (modulo T). To exclude loops we append clauses like

$$\neg 1!1{*} \vee \neg 2!1{*} \vee \neg 31!{*} \vee \neg 32!{*} \vee \neg 2!3{*} \vee \neg 22!{*} \vee \neg 1!2{*} \vee \neg 11!{*};$$

this one excludes the loop in the example illustration. There are 39 such loop-defeating clauses, one for each of the 13 simple cycles in each shadow.

Schaefer
Ben-Dor
Halevi
Mallach
CPLEX
GUROBI
Xeon computer
langford
per minute
Jabbour
Lonlac
Saïs
Salhi
exclusion clauses
at-most-one

Finally we can break symmetry by asserting the unary clauses 121_{T-1}, 111_0, 112_1 without loss of generality, after verifying that no solution can avoid all eight corners.

Clearly T must be an even number, because the graph is bipartite; also $T < 27$. If the method of exercise 12 is used for the exclusions, we obtain a total of 6264 clauses, 822 variables, and 17439 cells when $T = 16$; there are 9456 clauses, 1242 variables, and 26199 cells when $T = 24$. These clauses are too difficult for Algorithm D. But Algorithm L resolves them almost instantaneously for any given T; they turn out to be satisfiable if and only if $T = 24$, and in that case there are two essentially different solutions. One of these cycles, due to John Rickard (who introduced this problem at Cambridge University, circa 1990), is beautifully symmetric, and it is illustrated on the cover of Peter Winkler's book *Mathematical Mind-Benders* (2007). It can be represented by the delta sequence ($322\bar{3}133\bar{1}\bar{2}112\bar{3}\bar{2}23\bar{1}\bar{3}312\bar{1}\bar{1}\bar{2}$), where '$k$' and '$\bar{k}$' change coordinate k by $+1$ or -1. The other is unsymmetric and represented by ($3321\bar{2}1\bar{3}\bar{3}\bar{1}221\bar{2}323\bar{1}\bar{1}\bar{3}1\bar{2}\bar{1}\bar{3}\bar{2}$).

523. (Solution by Peter Winkler.) With coordinates (x, y, z) for $1 \le x \le m$, $1 \le y \le n$, $1 \le z \le 2$, any cycle with loopless shadows must contain at least two steps $(x, y, 1)$ — $(x, y, 2)$ and $(x', y', 1)$ — $(x', y', 2)$. We can assume that $x < x'$ and that $x' - x$ is minimum. The $m \times 2$ shadow contains $(x, 1)$ — $(x, 2)$ and $(x', 1)$ — $(x', 2)$, together with (say) the path $(x, 1)$ — $\cdots$ — $(x', 1)$, but without the edge $(x'', 2)$ — $(x''+1, 2)$ for some x'' with $x \le x'' < x'$. The unique shortest path from (x, y) to (x', y') in the $m \times n$ shadow contains some edge (x'', y'') — $(x''+1, y'')$; hence $(x'', y'', 1)$ — $(x''+1, y'', 1)$ must occur twice in the cycle.

524. This problem involves clauses very much like those for a cyclic path, but simpler; we have $T = 27$ and no "wrap-around" conditions. With typically 1413 variables, 10410 clauses, and 28701 cells, Algorithm L shines again, needing only a gigamem or two to handle each of several cases that break symmetry based on starting and ending points. There are four essentially different solutions, each of which can be assumed to start at 111; one ends at 333, another at 133, another at 113, and the other at 223. Using the delta sequence notation above, they are: $332\bar{3}\bar{3}2331\bar{3}\bar{3}\bar{2}33\bar{2}\bar{3}\bar{3}1332\bar{3}\bar{3}233$ (which is reflected ternary code); $31\bar{3}133\bar{1}\bar{1}211\bar{3}\bar{3}\bar{1}3\bar{1}\bar{3}231\bar{3}133\bar{1}\bar{1}$; $32\bar{3}231\bar{3}\bar{2}3\bar{2}\bar{3}132\bar{3}233\bar{2}\bar{2}\bar{1}22\bar{1}\bar{2}\bar{2}$; $1122\bar{1}\bar{1}\bar{2}13\bar{1}211\bar{2}\bar{2}\bar{1}\bar{1}31122\bar{1}\bar{1}\bar{2}1$.

[Such paths, and more generally spanning trees that have loopless shadows, were invented in 1983 by Oskar van Deventer, who called them "hollow mazes"; see *The Mathemagician and Pied Puzzler* (1999), 213–218. His *Mysterians* puzzle is based on an amazing Hamiltonian path on $P_5 \,\square\, P_5 \,\square\, P_5$ that has loopless shadows.]

525. The author's best solution, as of July 2015, had 100 variables, 400 clauses, and 1200 literals (cells); it was derived from Tseytin's examples of exercise 245, applied to a more-or-less random 4-regular graph of girth 6 on 50 vertices. Tseytin's construction, with one odd vertex and 49 even ones, yields 400 clauses of 4SAT, which are quite challenging indeed. It can be simplified to a 3SAT problem by insisting further that every even vertex must have degree exactly 2 in the subgraph specified by true edges. (See K. Markström, *J. Satisfiability, Boolean Modeling and Comp.* **2** (2006), 221–227).

That simplified problem still turned out to be fairly challenging: It was proved unsatisfiable by Algorithm L in 3.3 Tμ and by Algorithm C in 1.9 Tμ. (But by applying the endomorphisms of exercise 473, which broke symmetry by adding 142 clauses of length 6, the running time went down to just 263 Mμ and 949 Mμ, respectively.)

Another class of small-yet-difficult problems is worth mentioning, although it doesn't fit the specifications of this exercise [see I. Spence, *ACM J. Experimental Algorithmics* **20** (2015), 1.4:1–1.4:14]: Every instance of 3D matching whose representation

break symmetry
unary clauses
bipartite
exclusions
at-most-one
Rickard
Winkler
delta sequence
Winkler
break symmetry
reflected ternary code
spanning trees
van Deventer
hollow mazes
Mysterians
author
Tseytin
graph-based axioms
parity-related clauses
4-regular
girth
4SAT
Markström
comparison of run times
endomorphisms
broke symmetry
Spence
3D matching

as an exact cover problem has $5n$ rows and $3n$ columns, with five 1s in each column and three 1s in each row, can be represented as a SAT problem in $3n$ variables, $10n$ binary clauses, and $2n$ quinary clauses, hence only $30n$ total literals. This 5SAT problem has the same number of literals as the 3SAT problem discussed above, when $n = 40$; yet it is considerably more difficult if the matching problem is unsatisfiable. (On the other hand, the problem of this kind that defeated all the SAT solvers in the 2014 competition corresponds to a matching problem that is solved almost instantaneously by the dancing links method: Algorithm 7.2.2.1D needs less than 60 Mμ to prove it unsatisfiable.)

exact cover problem
5SAT
competition
dancing links

526. We prove by induction on $|F|$ that it's possible to leave at most $w(F)$ clauses unsatisfied, where $w(F) = \sum_{C\in F} 2^{-|C|}$: If all clauses of the multiset F are empty we have $w(F) = |F|$, and the result holds. Otherwise suppose the variable x appears in F. Let $l = x$ if $w(\{C \mid x \in C \in F\}) \geq w(\{C \mid \bar{x} \in C \in F\})$; otherwise $l = \bar{x}$. A simple calculation shows that $w(F \,|\, l) \leq w(F)$. [*JCSS* **9** (1974), 256–278, Theorem 3.]

999. ...

INDEX TO ALGORITHMS AND THEOREMS

Algorithm 7.2.2.2A, 28–29, 208.
Algorithm 7.2.2.2A*, 208–209.
Algorithm 7.2.2.2B, 31.
Lemma 7.2.2.2B, 58.
Theorem 7.2.2.2B, 59.
Algorithm 7.2.2.2C, 68.
Theorem 7.2.2.2C, 52–54.
Algorithm 7.2.2.2D, 33–34.
Algorithm 7.2.2.2E, 176.
Corollary 7.2.2.2E, 113.
Theorem 7.2.2.2E, 111.
Algorithm 7.2.2.2F, 205.
Theorem 7.2.2.2F, 86.
Theorem 7.2.2.2G, 70.
Algorithm 7.2.2.2I, 61.
Theorem 7.2.2.2J, 82.
Algorithm 7.2.2.2K, 216–217.
Theorem 7.2.2.2K, 90.
Algorithm 7.2.2.2L, 38–39.
Algorithm 7.2.2.2L$'$, 212.
Lemma 7.2.2.2L, 82.
Theorem 7.2.2.2L, 82.
Algorithm 7.2.2.2M, 82–83.
Theorem 7.2.2.2M, 83.
Algorithm 7.2.2.2P, 77.
Algorithm 7.2.2.2P$'$, 241.
Program 7.2.2.2P$'$, 242.
Algorithm 7.2.2.2R, 146.
Theorem 7.2.2.2R, 56.
Algorithm 7.2.2.2S, 93.
Theorem 7.2.2.2S, 87.
Algorithm 7.2.2.2T, 249–250.
Theorem 7.2.2.2U, 78–79.
Algorithm 7.2.2.2W, 79–80, 242.
Corollary 7.2.2.2W, 79.
Algorithm 7.2.2.2X, 44–45.

INDEX AND GLOSSARY

GOLDSMITH

The republic of letters is at present divided into three classes.
One writer, for instance, excels at a plan or a title-page,
another works away the body of the book,
and a third is a dab at an index.
— OLIVER GOLDSMITH, in *The Bee* (1759)

When an index entry refers to a page containing a relevant exercise, see also the *answer* to that exercise for further information. An answer page is not indexed here unless it refers to a topic not included in the statement of the exercise.

∂S (boundary set), 58, 154, 180, 188.
0–1 matrices, 106–109, 151, 176–177, 181, *see also* Grid patterns.
1SAT, 49, 148.
2-colorability of hypergraphs, 185.
2SAT, 49, 51–54, 77–78, 80, 101, 144, 147, 149, 157, 159, 266.
3-regular graphs, 147, 154, 231.
3CNF, 3, 148.
3D MATCHING problem, 134, 225, 290–291.
3D visualizations, 116–118.
3SAT, 3–4, 47–51, 59, 60, 78–80, 93–94, 131, 135, 146, 148–151, 153, 182–184, 231.
4-cycles, 109–110, 178, 225, 274.
4-regular graphs, 290.
4SAT, 49, 51, 150, 290.
5SAT, 51, 58, 224, 291.
6SAT, 51.
7SAT, 51, 151.
8 queens problem, 25, 282.
90°-rotational symmetry, 138, 202, 275.
100 test cases, 113–124, 127, 182, 184.
$\emptyset$ (the empty set), 185.
ϵ (the empty clause), 3, 27, 185, 291.
ϵ (the empty string), 3, 85.
ϵ (the tolerance for convergence), 93–94.
ε (offset in heuristic scores), 126, 213.
νx (1s count), *see* Sideways sum.
π (circle ratio), *see* Pi.
ρ (damping factor for variable activity), 67, 125–127, 155, 286.
ρ (damping factor for reinforcement), 93–94.
ϱ (damping factor for clause activity), 74, 125–127, 286.
τ parameter, 125–127, 235, 286.
$\tau(a,b)$ function, 147.
ϕ (golden ratio), 146, 147, 160, 251.
ψ (agility threshold), 76–77, 124–127, 240, 286.
ψ (confidence level), 93, 255.

a.s.: almost surely, 149, 153.
AAAI: American Association for Artificial Intelligence (founded in 1979); Association for the Advancement of Artificial Intelligence (since 2007), 67.
Absorbed clauses, 168.
Accordion solitaire, 282.
Achlioptas, Dimitris (Αχλιόπτας, Δημήτρης), 221.
`ACT(c)`, 74, 125.
`ACT(k)`, 66–68, 75, 125, 132.
Active path, 13.
Active ring, 32.
Activity scores, 67, 74–76, 125, 132, 155, 239.
Acyclic orientation, 161.
Adams, Douglas Noel (42), 126.
Adaptive control, 46, 126.
Addition, encoding of, 100–101, 114; *see also* Full adders, Half adders.
Adjacency matrix, 281.
Adjacent pairs of letters, avoiding, 248.
`AGILITY`, 76, 158, 240.
Agility level, 76, 124, 158.
Agility threshold (ψ), 76–77, 124–127, 240, 286.
Ahmed, Tanbir (তানবীর আহমেদ), 5, 147.
Alava, Mikko Juhani, 80.
Aldous, David John, 219.
Algorithm L^0, 39, 147.
Alice, 20–24, 139–141.
All-different constraint, 171.
All solutions, 143, 266.
Alon, Noga (נוגה אלון), 174, 254, 260.
Aloul, Fadi Ahmed (فادي أحمد العالول), 112, 281, 284.
Analysis of algorithms, 146–152, 158–160, 164.
Ancestors, 43.
AND operation, 9, 10, 13.
  bitwise ($x \,\&\, y$), 28, 29, 31, 37, 38, 66, 68, 76, 81, 196, 209–211, 220, 241.
André, Pascal, 131.
Anisimov, Anatoly Vasilievich (Анисимов, Анатолий Васильевич), 249.
Annexstein, Fred Saul, 274.
Anti-maximal-element clauses, 56, 62, 97, 115, 153, 155, 157, 167.

Antisymmetry, 178.
Appier dit Hanzelet, Jean, 57.
April Fool, 7.
Ardila Mantilla, Federico, 256.
Arithmetic progressions, 4, 114.
  avoiding, 135.
Armies of queens, 180.
Asín Achá, Roberto Javier, 267.
Asserting clause, *see* Forcing clause.
Associative block design, 4.
Associative law, 227.
Asymmetric Boolean functions, 178.
Asymmetric elimination, 260.
Asymmetric tautology, *see* Certifiable clauses.
Asymptotic methods, 53–54, 147–151, 164, 210, 226, 230, 283.
At-least-one constraint, 171, 265.
At-most-one constraint, 6, 97–99, 103, 104, 120, 134, 149, 170, 171, 238, 265, 266, 289.
ATPG: Automatic test pattern generation, *see* Fault testing.
Atserias, Albert Perí, 262.
Audemard, Gilles, 72.
Aurifeuille, Léon François Antoine, factors, 14.
Autarkies, 44, 71, 146, 152, 177, 214, 215, 217.
  testing for, 146, 214.
Autarky principle, 44.
Automatic test pattern generation, *see* Fault testing.
Automaton, 272.
Automorphisms, 108, 111, 180, 197, 236, 277.
Autosifting, 220.
Auxiliary variables, 6, 8, 15, 17, 60, 97, 101, 104, 105, 109, 135, 136, 148, 170–174, 186, 262, 268, 276–279, 280–281, 287.
`AVAIL` stack, 257.
Averages, 120.
Avoiding submatrices, 106–107.
Awkward trees, 227.
Axiom clauses, 54, 59, 100, 264, 266.

Bacchus, Fahiem, 73, 271.
Backjumping, 64, 68, 74, 132, 233, 236, 239.
Backtrack trees, *see* Search trees.
Backtracking, 4, 27–34, 38–39, 64, 105, 128, 129, 132, 151, 176, 190, 204, 219, 231, 236.
Bailleux, Olivier, 8, 26, 135, 137, 143, 174, 272.
Baker, Andrew Baer, 98.
Balas, Egon, 206.
Baldassi, Carlo, 93.
Ball, Walter William Rouse, 180.
Ballot numbers, 78.
Balls and urns, 221.
Banbara, Mutsunori (番原睦則), 264, 267, 268.
Bartley, William Warren, III, 129.
Basket weavers, 141.
Batcher, Kenneth Edward, 266.
Baumert, Leonard Daniel, 265.
Bayardo, Roberto Xavier, Jr., 132.
Bayes, Thomas, networks, 95.
BCP: Boolean constraint propagation, *see* Unit propagation.
BDD: A reduced, ordered binary decision diagram, 17–18, 102, 103, 132, 137, 148, 174, 181, 188, 193, 194, 197, 202, 220.
BDD base, 219.
Belief propagation, 95.
Ben-Dor, Amir (אמיר בן-דור), 289.
Ben-Sasson, Eli (אלי בן-ששון), 57–58, 153, 231.
Benchmark tests, 35, 131–133, 139, 147, 190, 206.
  100 test cases, 113–124, 127, 182, 184.
Bender, Edward Anton, 250.
Beresin, May, 275.
Berghammer, Rudolf, 204.
BerkMin solver, 132.
Berlekamp, Elwyn Ralph, 17.
Berman, Piotr, 224.
Bernhart, Frank Reiff, 188.
Bernoulli, Jacques (= Jakob = James), distribution, multivariate, 89.
Bethe, Hans Albrecht, 95.
Better reasons, 157.
Bias messages, 92.
Biased random bits, 12, 241.
Biere, Armin, v, 66, 76, 96, 129, 132, 166, 188, 258, 260, 261, 269, 280.
Big clauses, 145.
`BIMP` tables, 36–41, 43, 45, 124, 144, 235.
Binary addition, 114.
Binary clauses, 3, 6, 36, 124, 133, 155–156.
Binary constraints, 171.
Binary decoder, 179.
Binary implication graph, *see* Dependency digraph, 41.
Binary matrices, 106–109, 151, 176–177, 181, *see also* Grid patterns.
Binary multiplication, 8.
Binary number system, 9, 98.
Binary recurrence relations, 189.
Binary relations, 56.
Binary search, 187.
Binary strings, 181.
Binary tensor contingency problem, 142, 151.
Binomial coefficients, 149.
Binomial convolutions, 250.
Bipartite graphs, 58, 177, 290.
Bipartite matching, 150.

Bipartite structure, 90.
Birthday paradox, 49.
Bishops, 141.
Bitmaps, 17, 139.
Bitwise operations, 11, 12, 81, 158, 161, 241, 246, 258–259.
Black and blue principle, 146, 216.
Black and white principle, 146.
Blake, Archie, 130.
blit, 234, 236.
Block decomposition, 275.
Block designs, 106.
Block diagonal matrices, 177.
Blocked clauses, 102, 215, 260, 261, 269.
  binary, 146.
  elimination of, 167.
Blocked self-subsumption, 167.
Blocking digraph, 215.
Blocks in Life, 197, 200.
Bloom, Burton Howard, coding, 258.
Bloom, Thomas Frederick, 185.
Bob, 20–24, 115, 139–141.
Böhm, Max Joachim, 131.
Bollobás, Béla, 54, 220.
Bonacina, Maria Paola, 129.
book graphs, 126, 179.
Boole, George, 129.
Boolean chains, 9, 11, 12, 102, 114, 173.
  optimum, 178–179.
Boolean formulas, 1.
Boolean functions, 14–16.
  expressible in kCNF, 220.
  synthesis of, 178–179.
Boppana, Ravi Babu, 174.
Borgs, Christian, 54.
Bottom-up algorithms, 252.
Boufkhad, Yacine (ياسين بوفخد), 8, 26, 131, 135, 137, 143, 174, 272.
Boundary sets, 58, 154, 180, 188.
Boundary variables, 230.
Bounded model checking, 16–24, 132, 137–141, 157, 179–180.
Branching heuristics, 105, 144, *see also* Decision literals.
Branching programs, 102, 173, 174.
Branchless computation, 242.
Braunstein, Alfredo, 90, 91, 256.
Breadth-first search, 37, 43, 68, 130, 235.
Break count, 79.
Breaking symmetries, vii, 5, 19, 105–114, 138, 176–181, 187, 188, 190–192, 238, 267, 281–283, 285, 288–290.
  in graph coloring, 99–100, 114, 171, 179, 187.
Broadcasting, 170.
Broadword computations, 11, 12, 158, 161, 246, 258.
Brown, Cynthia Ann Blocher, 30, 32, 131, 151, 226.
Brown, Thomas Craig, 185.
Brummayer, Robert Daniel, 269.
Brunetti, Sara, 206.
Bryant, Randal Everitt, v, 7, 187.
BST(l), 211.
BSTAMP counter, 211.
Buckingham, David John, 197, 200.
Buddy system, 36, 144, 235.
Bugrara, Khaled Mohamed (خالد محمد ابوغراره), 226.
Bugs, 16, 69, 77, 133, 240.
Bulnes-Rozas, Juan Bautista, 215.
Bumped process, 21.
Bundala, Daniel, 196.
Burney, Charles, viii.
Burns, James Edward, 204.
Buro, Michael, 131.
Buss, Samuel Rudolph, v, 153, 270.
Bystanders, *see* Easy clauses.

C-SAT solver, 131.
Cache memories, 24.
Calabro, Chris, 288.
Candidate variables, 40–44, 131, 214.
Canonical forms, 138, 248.
Cardinality constraints, 7–8, 26, 104, 106, 113, 114, 121, 135, 143, 187, 188, 193, 194, 196, 204, 285.
  for intervals, 100, 190, 280.
Carlier, Jacques, 131.
Carlitz, Leonard, 162.
Carriers in Life, 197, 200.
Carroll, Lewis (= Dodgson, Charles Lutwidge), 129–130.
Carry bits, 9, 12, 101, 192, 193.
Cartier, Pierre Emile, 83, 86, 163.
Case analysis, 27, 130.
CDCL (conflict driven clause learning) solvers, 62–71, 103, 121, 132–133, 155.
  combined with lookahead solvers, 129.
  compared to lookahead solvers, 98–100, 118–121, 182, 290.
Cells of memory, 28, 122–124.
Cellular automata, 17, 202.
Certifiable clauses, 168, 260.
Certificates of unsatisfiability, 69–71, 157, 169, 176, 178.
Chaff solver, 67, 132.
Chain rule for conditional probability, 254.
Chains, *see* Boolean chains, Resolution chains, s-chains.
Channel assignment, 136.
Channeling clauses, 264.
Characteristic polynomial of a matrix, 163, 218.
Chavas, Joël, 91.
Chayes, Jennifer Tour, 54.
Chebyshev (= Tschebyscheff), Pafnutii Lvovich (Чебышевъ, Пафнутій

Львовичъ = Чебышев, Пафнутий Львович), inequality, 221.
polynomials, 247.
Cheshire Tom, 24–26, 115, 142–143.
Chess, 7, 170.
Chessboards, 18, 25, 99, 106, 115, 138, 180.
Chiral symmetry (rotation but not reflection), 138, 202, 275.
Chordal graphs, 163–164.
Chromatic number $\chi(G)$, 99, 135–136, 147, 174, 281.
Chung Graham, Fan Rong King (鍾金芳蓉), 283.
Chvátal, Václav (= Vašek), 5, 52, 59, 185.
Cimatti, Alessandro, 132.
Circuits, Boolean, 10, 101–103, 114, *see also* Boolean chains.
Circular lists, 32.
Clarke, Edmund Melson, Jr., 132.
Clashing pairs of letters, 84.
Clausal proofs, *see* Certificates of unsatisfiability.
Clause activity scores, 74, 239.
Clause-learning algorithms, 61–62, 103, 118, 121, 132–133, 154–155.
Clauses per literal, 150, 231; *see also* Density of clauses.
Claw graph, 249.
Clichés, 76.
Clique Local Lemma, 165.
Cliques, 81, 100, 162, 167, 169, 171, 179, 190.
covering by, 165.
Closest strings, 114, 181, 182.
Clusters, 166.
CNF: Conjunctive normal form, 9, 101, 154, 173, 193, 196.
Cocomparability graphs, 249, 250.
Coe, Timothy Vance, 201.
Coexisting armies of queens, 180.
Cographs, 163, 250.
Cohen, Bram, 79, 246.
Coja-Oghlan, Amin, 221.
Colexicographic order, 206, 278.
Coloring a graph, 6–7, 99–100, 153, 179, 260.
fractional, 135.
multiple, 135.
of queens, 99–100, 114–115, 171.
radio, 136.
Column sums, 151.
Commutative law, 27, 180, 227.
partial, 83, 250–251.
Comparator modules, 115, 137.
Comparison, lexicographic, 101, 111–113.
Comparison of running times, 34–35, 39, 69, 97–100, 105–107, 110, 112, 118–128, 182, 184, 218, 237, 264, 281, 290.
Compensation resolvents, 39, 144, 147.
Competitions, 131–133, 291.
Complement of a graph, 134.
Complementation of unary representations, 100.
Complemented literals, 2–4, 37, 62–64, 78, 111, 210, 266.
Complete binary trees, 8, 135, 230.
Complete bipartite graphs $K_{m,n}$, 250, 254.
Complete graphs K_n, 153, 178, 186, 262.
Complete k-partite graphs, 250, 262.
Complete t-ary trees, 160.
Compressing, *see* Purging unhelpful clauses.
Conditional autarkies, 215.
Conditional expectation inequality, 150.
Conditional symmetries, 107, *see* Endomorphisms.
Conditioning operations ($F\,|\,l$ and $F\,|\,L$), 27, 96, 143, 157, *see* Unit conditioning.
Cones in trace theory, 87.
Confidence level (ψ), 93, 255.
Conflict clauses, 63, 70, 171; *see also* Preclusion clauses.
Conflict driven clause learning, 62–69, 103, 121, 132–133, 155.
Conflicts, 62, 124, 132.
Conjunctive normal form, 1, 9, 101, 154, 173, 193, 196.
irredundant, 257.
Conjunctive prime form, 104.
Connected graphs, 177.
Connectedness testing, 169–170.
Connection puzzles, 170.
CoNP-complete problems, 3, 207.
Consecutive 1s, 88, 175, 254.
Consensus of implicants, 130.
Consistent Boolean formulas, *see* Satisfiable formulas.
Consistent partial assignments, 30, 165.
Constrained variables in partial assignments, 165–166.
Contests, 131–132.
Context free languages, 175.
Contiguous United States, 136.
Contingency tables, binary, 142.
3D, 151.
Convex functions, 216.
Convex hulls, 247.
Convolution principle, 250.
Conway, John Horton, 17, 139, 201.
Cook, Stephen Arthur, 61, 62, 130–131, 154, 229, 237.
cook clauses, 157.
Cooper, Alec Steven, 285.
Core assignments, 166.
Core of Horn clauses, 174, 216.
Coupon collector's test, 220.
Covering assignments, 166, 221, 255.
Covering problems, 2, 193, 194, *see also* Domino coverings.
Covering strings, 181.

CPLEX system, 26, 289.
CPU: Central Processing Unit (one computer thread), 121.
Crawford, James Melton, Jr., 98, 113.
Cray 2 computer, 137.
Critical sections, 21–23, 140–141.
Crossover point, *see* Threshold of satisfiability.
Crusoe (= Kreutznaer), Robinson, vii.
CSP: The constraint satisfaction problem, 283.
Cube and conquer method, 129.
Cubic graphs (3-regular, trivalent), 147, 154, 231.
Cufflink pattern, 255.
Culver, Clayton Lee, 185.
Cut rule, 59.
Cutoff parameters, 41, 145.
Cutting planes, 184, 206.
Cycle detection problem, 260.
Cycle graphs C_n, 135, 160, 262.
Cycle structure of a permutation, 108, 112–113, 163, 178, 277.
Cyclic DPLL algorithm, 33.
Cyclic patterns, 19.
Cyclic permutations, 163.

da Vinci, Leonardo di ser Piero, 7.
Dadda, Luigi, 9, 114, 136, 173.
Dags: Directed acyclic graphs, 54.
of resolutions, 54–56, 70.
Damping factors, 46, 67, 74, 76, 93–94, 125, 126, 155.
Dancing links, 5, 121, 134, 208, 288, 291.
Dantchev, Stefan Stoyanov (Данчев, Стефан Стоянов), 110.
Darwiche, Adnan Youssef (عدنان يوسف درويش), 67, 262.
Data structures, 28–34, 36–38, 43, 66–67, 80, 95–96, 143–145, 155–156, 159, 167, 238, 273.
Davis, Martin David, 9, 31–32, 130, 298.
Dawson, Thomas Rayner, 170.
De Morgan, Augustus, laws, 3.
de Vries, Sven, 206.
de Wilde, Boris, 213.
Deadlock, 22–23.
Debugging, 69, 77.
Dechter, Rina Kahana (רינה כהנא דכטר), 67.
Decision literals, 62, 69, 124, 132.
Decision trees, *see* Search trees.
Decomposable matrices, 177.
Default parameters, 93, 125–126.
Default values of gates, 11.
Definite Horn clauses, 174.
Defoe, Daniel (= Daniel Foe), vii.
Degree of a vertex, 191.
Degrees of truth, 37–39, 42–43, 45–46, 216.
Dekker, Theodorus Jozef, 140.
Del Lungo, Alberto, 206.
Delayer, 55–56, 152–153.
Deletion from a heap, 234.
Delta sequence, 290.
Demenkov, Evgeny Alexandrovich (Деменков, Евгений Александрович), 280.
Density of clauses: The number of clauses per variable, 50–51, 150, 231, 288.
Dependence graph in trace theory, 248.
Dependence of literals, 63.
Dependency digraph (of literals), 41, 131, 168, 215, 237, 260.
Dependency-directed backtracking, *see* Backjumping.
Dependency graph (of events), 82, 164, 165.
Dependency on a variable, 137.
Depth-first search, 130.
Dequen, Gilles Maurice Marceau, 131.
Determinants, 162, 163, 251.
Deterministic algorithm, 17, 120.
Deventer, Mattijs Oskar van, 290.
DFAIL, 46, 147.
Dfalse literals, 45.
Diagonals of a matrix, 24–25, 141–142.
Diagram of a trace, 84.
Díaz Cort, José Maria (= Josep), 51.
Dick, William Brisbane, 180.
Difficult instances of SAT, 5, 14, 26, 51, 55–59, 118–121, 153–154, 184, 190, 192, 197, 206, 280.
Digital tomography, 24–26, 115, 141–143, 167, 285.
Digraphs, 54, 108, 161, 162, 263, *see also* Blocking digraph, Dependency digraph, Implication digraph.
Dijkstra, Edsger Wybe, 22, 202, 204.
DIMACS: Center for Discrete Mathematics and Theoretical Computer Science, 131.
DIMACS: *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, inaugurated in 1990.
Ding, Jian (丁剑), 51.
Direct encoding, 98, 114, 171, 186, 264, 265, 281.
Direct sum of graphs or matrices, 162, 177.
Directed acyclic graphs of resolutions, 54–56, 70.
Directed graphs, *see* Digraphs.
Discarding the previous learned clause, 72, 156.
Discrepancy patterns, 114, 182.
Disjoint shortest paths, 276.
Disjunctive normal forms, 14, 115, 130, 195, 257.
Distance $d(u,v)$ in a graph, 262.
Distinct literals, 2.
Division of traces, 85, 161, 250.

DNF: Disjunctive normal form, 14, 115, 130, 195, 257.
Dodgson, Charles Lutwidge, 129–130.
Domino coverings, 110, 114, 115, 143, 177, 178.
Don't-cares, 194, 280.
Double clique hints, 100, 114, 171.
Double coloring, 115, 135.
Double lookahead, 45–46, 126, 131, 286.
Double order, 214.
Double truth, 45.
Doubly linked lists, 28, 257, 259.
Downhill resolution, 96, 166.
Downhill transformations, 95.
Doyle, Arthur Ignatius Conan, 72.
DPLL (Davis, Putnam, Logemann, Loveland) algorithm, 32–33, 62.
  with lookahead, 38, 131.
DT (double truth), 45.
Dtrue literals, 45.
Dual of a Boolean function, 130, 174.
Dubois, Olivier, 131.
Dudeney, Henry Ernest, 114, 263.
Dufour, Mark, 37.
Dull, Brutus Cyclops, 181.
Durfee, William Pitt, square, 276.
Dynamic storage allocation, 144.
Dynamical system, discrete, 16.

e, as source of "random" data, 12, 193.
Eager data structures, 30, 36, 156.
Easy clauses, 149.
Eaters in Life, 20, 139.
Eén, Niklas Göran, v, 67, 96, 166, 203, 260, 268.
Ehlers, Thorsten, 196.
Eightfold symmetry, 138, 198.
Elegance, 35–36, 196.
Elimination of clauses, 167–168; *see also* Purging unhelpful clauses.
Elimination of variables, 60–61, 95–97, 101, 102, 129, 130, 154–155, 166–168, 173, 174, 256–257, 259–260, 270, 272.
Embedded graphs, 169, 262.
Empilements, 84, 161, 248.
Empirical performance measurements, 122–124.
Empty clause (ϵ), 3, 27, 185.
Empty list, representation of, 33, 210.
Empty partial assignment, 166.
Empty set ($\emptyset$), 185.
Empty string (ϵ), 3, 85.
Encoding into clauses, 6, 18, 97–105, 120, 134, 170, 179, 198, 202.
  ternary data, 100, 141, 179.
Endomorphisms, 107–111, 177–178, 181, 281, 290.
Equal sums, encoding of, 174.
Equally spaced 1s, 4, 114, 135; *see also waerden*.

Equivalence classes in trace theory, 84.
Equivalence of Boolean functions, 178.
Erdős, Pál (= Paul), 81, 107, 190, 281.
  discrepancy patterns, 114, 179, 182.
Erp rules, 95–96, 166–168, 259.
Evaluation of Boolean functions, 137, 178–179, 194.
Even-length cycles, 277.
Even-odd endomorphisms, 110, 177–178.
Exact cover problems, vii, 2, 5–6, 28, 134, 183, 186, 219, 225, 257, 291.
  by pairs (perfect matchings), 109–110, *see also* Domino coverings.
  by triples (3D MATCHING), 134, 225, 290–291.
  fractional, 135–136.
Exclusion clauses, 6, 21, 99, 114, 134, 149, 153, 238, 260, 289.
Exclusive or, ternary, 136.
Existential quantifiers, 60.
Expander graphs, 58, 231.
Exploitation stack, 259.
Exploration phase of lookahead, 40, 43–44.
Exponential time, 144.
  hypothesis, 288.
Extended resolution, 60, 71, 133, 154, 168, 215.
Extreme distribution, 87, 89, 163.

factor_fifo(m, n, z), 10, 12, 114, 184, 192.
factor_lifo(m, n, z), 10, 114, 184, 192.
factor_rand(m, n, z, s), 10, 184.
Factorization, 8–10, 136, 184, 192.
  of traces, 86, 162, 250.
Failed literals, 97, 167, 175, 269.
Fallacious reasoning, 16, 284.
False hits, 258.
False literals preferred, 31, 33, 67, 125–127, 286.
Fanout gates, 10–14, 136.
Fat clauses, 58.
Fault testing, 10–14, 114, 126, 136–137, 167, 260.
Feedback mechanism, 46, 104.
Fermat, Pierre de, 10.
Fernandez de la Vega, Wenceslas, 52.
Fibonacci, Leonardo, of Pisa (= Leonardo filio Bonacii Pisano), numbers, 160, 215, 254.
  ruler function, 246.
Fichte, Johannes Klaus, 262.
Field of a variable, 91, 165.
FIFO: first in, first out, 10.
Finite-state automata, 175.
First in, first out, 10.
First moment principle, 53, 148, 150.
First order logic, 59, 130.
Fischetti, Matteo, 206.
Fixed point of endomorphisms, 177.

Fixed values of literals, 37, 42–46.
FKG inequality, 89.
Flag bits, 235.
Flammenkamp, Achim, 198.
Flexibility coefficients, 91.
Flickering state variables, 141.
Flipflops in Life, 138, 143.
Floating point arithmetic, 91–92, 217, 239.
  overflow, 67.
Floor tiling, 115, 143, 199.
Flower snarks, 69, 147, 153, 157.
Flushing literals and restarting, 68, 75–77, 124, 132, 157, 158, 169, 234, 246.
Foata, Dominique Cyprien, 83, 86, 163.
Focus of attention, 41, 67, 91, 132.
Foe, Daniel (= Daniel Defoe), vii.
Footprint heuristic, 279.
Forced literals, 45.
Forcing clause, 62, *see* Unit propagation.
Forcing representations, 104–105, 174, 175, 274.
Forests, 43, 87, 163.
Forgetting clauses, 168, 169, *see* Purging unhelpful clauses.
Four bit protocol, 115.
Four Color Theorem, 7.
Fourfold symmetry, 138.
Fractional coloring number, 135.
Fractional exact cover, 135–136.
Frances, Moti (מודי פרנסס), 285.
Franco, John Vincent, 131, 148, 225, 274.
Free literals and free variables, 38, 66, 165–166.
Freeman, Jon William, 131.
Friedgut, Ehud (אהוד פרידגוט), 51.
Frontier ($\partial_{\rm in}$), 180, 188.
Frost, Daniel Hunter, 67.
fsnark clauses, 69, 71, 114, 147–148, 153, 157.
Full adders, 9, 136, 179, 192, 268, 278.
  modified, 114, 280.
Full runs, 73, 158, 235.
Furtlehner, Cyril, 91.

Gμ: Gigamems = billions of memory accesses, 97, 118.
  per minute, 289.
Gadgets, 134, 183.
Gallager, Robert Gray, 95.
Game of Life, 17–20, 97, 114, 137–139, 143, 167.
Garden of Eden, 139.
Gardner, Martin, 7, 19, 181, 188, 283.
Gates, 10–13, 101–103, 121, 136.
GB_GATES, 13–14.
Gebauer, Heidi Maria, 224.
Geek art, 116–117.
Generalization of resolution, 226.
Generating functions, 85, 89–90, 151, 158–159, 164, 188, 194, 219, 222, 230, 284.
  exponential, 162.
Generic graph, 174.
Gent, Ian Philip, v, 265, 283.
Gentzen, Gerhard Karl Erich, 59.
Geometric distribution, 244, 262.
Georges, John Pericles, 192.
Gerdes, Paulus Pierre Joseph, 205.
Gessel, Ira Martin, 248.
Giant strong component, 52.
Gigamem (Gμ): One billion memory accesses, 35, 39, 97, 188.
  per minute, 289.
Ginsberg, Matthew Leigh, 113.
Gipatsi patterns, 141.
Girth of a graph, 176, 290.
Given literals ($F\,|\,l$ or $F\,|\,L$), 27, 96, 143, 157; *see also* Unit conditioning.
Gliders in Life, 19, 138–139, 197, 201.
  symmetry of, 200.
Global ordering, 284.
Glucose measure, *see* Literal block distance.
Goerdt, Andreas, 52.
Goldberg, Allen Terry, 131, 225.
Goldberg, Eugene Isaacovich (Гольдберг, Евгений Исаакович), 70, 132.
Goldman, Jay Robert, 250.
Goldsmith, Oliver, 293.
Golomb, Solomon Wolf, 265.
González-Arce, Teófilo Francisco, 268.
Gosper, Ralph William, Jr., 20, 198.
Goultiaeva, Alexandra Borisovna (Гультяева, Александра Борисовна), 73.
Grabarchuk, Peter Serhiyevich (Грабарчук, Петро Сергійович), 263.
Grabarchuk, Serhiy Alexeevich (Грабарчук, Сергій Олексійович), 263.
Grabarchuk, Serhiy Serhiyevich (Грабарчук, Сергій Сергійович), 263.
Graham, Ronald Lewis (葛立恆), 185, 283.
Graph-based axioms, 59, 154, 178, 290.
Graph coloring problems, *see* Coloring a graph.
Graph embedding, 169, 262.
Graph layout, 116–118.
Graph quenching, 114, 179–180, 281.
Gray, Frank, codes, 201, 282.
Greedy algorithms, 80, 136, 172.
Greenbaum, Steven Fine, 102.
Grid graphs, 110, 136, 151, 162–163.
  list coloring of, 151.
Grid patterns, 17–20, 24–26, 137–139, 142.
  rotated 45°, 141–142.
Griggs, Jerrold Robinson, 192.
Grinbergs, Emanuels Donats Frīdrihs Jānis, 218.

Gritzman, Peter, 206.
Grötschel, Martin, 264.
Gu, Jun (顾钧), 77.
Guéret-Jussien, Christelle, 267.
Guilherme De Carvalho Resende, Mauricio, 16.
GUROBI system, 289.
Guy, Richard Kenneth, 17, 19, 107.
Gwynne, Matthew Simon, 105, 270, 273.

Hackers, 199.
Haken, Armin, 57, 58.
Halevi, Shai (שי הלוי), 289.
Half adders, 9, 192, 268.
Halting problem, 130.
Hamadi, Youssef (ⵃⴰⵎⴰⴷⵉ ⵢⵓⵙⴻⴼ, حمادي يوسف), 236.
Hamilton, William Rowan, cycles, 169.
  paths, 184.
Hamming, Richard Wesley, distance, 285.
Han, Hyojung (한효정), 236.
Handwaving, 89.
Hanzelet, *see* Appier dit Hanzelet.
Hard clauses, 168.
Hard sudoku, 183.
Hartman, Christiaan, 202.
Haven, G Neil, 131.
Head of the list, 28.
Header elements, 225.
`HEAP` array, 67, 158, 233–235, 240.
Heap data structure, 67, 214.
  insertion and deletion from, 234.
Heaps of pieces, 83.
Height of a literal, 214.
Height of a trace, 85.
Heilmann, Ole Jan, 251.
Helpful rounds, 169.
Heule, Marienus (= Marijn) Johannes Hendrikus, iv, v, 37, 40, 46, 71, 75, 97–98, 104, 129, 134, 147, 182, 186, 202, 213, 239, 260, 261, 263.
Heuristic scores, 90–95.
  for clauses, 72–74, 125–127, 158, 239, 286.
  for variables, 40–44, 61, 67, 80, 126, 145–147, 214.
Hidden weighted bit function, 173.
Hierarchy of hardness, 176, 178.
Hilton, Anthony John William, 191.
Hint clauses, 100, 114, 171.
Hirsch, Edward Alekseevich (Гирш, Эдуард Алексеевич), 215.
Historical notes, 32, 59–60, 105, 129–133, 231.
Hollow mazes, 290.
Holmes, Thomas Sherlock Scott, 72.
Homomorphic embedding, 169, 262.
Honest representations, 105, 270.
Hoory, Shlomo (שלמה חורי), 224.
Hoos, Holger Hendrik, v, 125–127, 133, 160.
Horn, Alfred, clauses, 132, 166, 176, 216, 263.
  core of, 174, 216.
  renamed, 176, 263.
Horsley, Daniel James, 274.
Horton, Robert Elmer, numbers, 152.
Hsiang, Jieh (項傑), 129.
Hume, David, 1.
Hunt, Warren Alva, Jr., 71, 239.
Hutter, Frank Roman, 125, 133.
Hypergraph 2-colorability, 185.
Hyperresolution, 56, 228, 257.

Idle Year solitaire, 180.
If-then-else operation (u? v: w), 81, 102, 152, 173, 219.
ILS: Iterated Local search, 125.
Impagliazzo, Russell Graham, 55, 288.
Implication digraph, 52, 144.
in situ deletion, 156.
Inclusion and exclusion, 221–222, 256.
Inconsistent clauses, *see* Unsatisfiable formulas.
Indecomposable matrices, 177.
Independent vertices, 7, 147.
Induced graph, 262.
Induction proofs by machine, 24, 203.
Infinite loop, 244.
Initial guess for literals, 31, 33, 66, 125–127, 286.
Initial state X_0, 16–17, 21, 24, 140, 202.
Inprocessing, 95, 168.
Input and output, 120.
Input states, 175.
Insertion into a heap, 234.
Integer programming, 26, 184, 206, 285.
Interactive SAT solving, 142.
Interlaced roots of polynomials, 163.
Internet, ii, iii, v, 118.
Intersection graphs, 84, 161.
Interval graphs, 87, 163.
Intervals, cardinality constrained to, 100, 190, 280.
Invariant assertions, 23–24, 43, 115, 140, 203, 216, 217, 255, 261.
Inverse permutations, 112, 265.
Inversions of a permutation, 213.
Involution polynomial of a set, 163.
Involutions, signed, 112–113, 180, 277–278.
`INX` array, 38, 211, 286.
IP: Integer programming, 26, 184, 206, 285.
Irredundant CNF, 257.
Irreflexive relation, 56.
Irving, Robert Wylie, 151.
Isaacs, Rufus Philip, 218.
Isolated vertices, 262.
`IST` array, 38.
`ISTACK` array, 38, 145.
`ISTAMP` counter, 37–38, 46, 145.

Iterated local search, 125.
Iwama, Kazuo (岩間一雄), 224.

Jabbour, Saïd (ⵊⴰⴱⵓⵔ ⵙⴰⵄⴷ, سعيد جبور), 236, 289.
Jacquet, Philippe Pierre, 225.
Jagger, Michael Philip “Mick”, 1.
Janson, Carl Svante, v.
Järvisalo, Matti Juhani, 105, 132, 260, 261.
Jeannicot, Serge, 215.
Jeavons, Peter George, v.
Jerrum, Mark Richard, 151.
Job shop problems, 172.
Johnson, David Stifler, 184, 191.
Johnson, Samuel, viii.
Join of graphs, 162.

k-induction, 203.
Kμ: Kilomems = thousands of memory accesses, 98.
$K_{m,n}$ (complete bipartite graph), 176.
Kamath, Anil Prabhakar (अनिल प्रभाकर कामथ), 16.
Kaporis, Alexis Constantine Flora (Καπόρης, Αλέξιος Κωνσταντίνου Φλώρας), 51.
Karmarkar, Narendra Krishna (नरेन्द्र कृष्ण करमरकर), 16.
Karp, Richard Manning, 52, 268.
Karpiński (= Karpinski), Marek Mieczysław, 224.
Karpovsky, Mark Girsh, 285.
Kasif, Simon (שמעון כסיף), 265.
Katona, Gyula (Optimális Halmaz), 107.
Kautz, Henry Alexander, 79, 132.
Kaye, Richard William, 207.
Keller, Robert Marion, 83.
Kernel of a graph, 99, 134, 186, 188, 218.
  clauses for, 114, 134.
Kilomem (Kμ): One thousand memory accesses, 39, 98.
Kim, Jeong Han (김정한), 54.
King moves, 134, 169.
Kingwise connected cells, 170.
Kirousis, Lefteris Miltiades (Κυρούσης, Ελευθέριος Μιλτιάδης), 51.
Kitagawa, Satoshi (北川哲), 264, 267.
Kleine Büning (= Kleine-Büning), Hans Gerhard, 131, 185.
Knapsack problem with a partial ordering, 158.
Knessl, Charles, 225.
Knight moves, 115, 169.
Knuth, Donald Ervin (高德纳), i, vi, 1, 14, 16, 19, 51–52, 72, 74, 93, 94, 118, 125–127, 192, 193, 195, 197, 202, 210, 212, 213, 216, 227, 235–237, 240, 242, 249, 260, 264–267, 275, 278, 279, 282, 286, 290.
Knuth, John Martin (高小强), *see* Truth.
Kojevnikov, Arist Alexandrovich (Кожевников, Арист Александрович), 280.
Kolipaka, Kashyap Babu Rao (కొలిపాక కశ్యప్ బాబు రావు), 90, 161, 255.
Konev, Boris Yurevich (Конев, Борис Юрьевич), 281.
Kouřil, Michal, 5, 185.
Kroening, Daniel Heinrich Friedrich Emil, v, 203, 288.
Krom, Melven Robert, clauses, *see* 2SAT.
kSAT, 3, 49–51, 146, 148, 150, 183.
Kulikov, Alexander Sergeevich (Куликов, Александр Сергеевич), 280.
Kullmann, Oliver, v, 5, 105, 129, 147, 152, 215, 216, 218, 228, 260, 270, 273.
Kwekkeboom, Cornelis (= Kees) Samuël, 202.

$L(2,1)$ labeling of graphs, 136.
L_7 lattice, 255.
Labeled pyramids, 162.
Labeled traces, 162.
Lalas, Efthimios George (Λάλας, Ευθύμιος Γεωργίου), 51.
Lamport, Leslie B., 24, 204.
Land mines, 142.
Landman, Bruce Michael, 185.
Langford, Charles Dudley, problem of pairs, vii, 5–6, 34, 98, 121, 125, 134, 170, 186, 289.
langford(n), 6, 34–35, 39, 97, 98, 114, 121, 134, 210, 236, 289.
langford$'(n)$, 6, 98, 114, 134, 289.
langford$''(n)$, 98.
langford$'''(n)$, 264.
Larrabee, Tracy Lynn, 13, 137.
Las Vegas algorithms, 159–160.
Last in, first out, 10.
Late Binding Solitaire, 114, 180.
Latin rectangle construction, 151.
Lattices of partial assignments, 165–166.
Lauria, Massimo, v, 56.
Laurier, Jean-Louis, 187.
Lazy data structures, 30–34, 36, 65, 156, 234.
Le Berre, Daniel Claude Yves, 132.
Learned clauses, 63–65, 70–71, 124, 132, 168.
  sequence of, 70, 156.
Learning a Boolean function, 14–16, 115, 137.
Least common ancestor, 253.
Left division of traces, 85, 161.
Left factor of a trace, 161–162.
Lemaire, Bernard François Camille, 282.
Lemma generation, *see* Clause-learning algorithms.
Length of a trace, 85.
Lettmann, Theodor August, 185.
Level 0, 62, 66, 124, 156, 207, 233.

Levels of values, 62–66, 156, 233.
Levesque, Hector Joseph, 50.
Levine, Eugene, 275.
Lewis, Jerome Luther, 275.
Lex-leader: The lexicographically smallest element, 111, 283.
Lexicographic order, 4, 25, 26, 30, 101, 105, 107, 109, 111–113, 115, 197, 282–283.
  encoded in clauses, 101, 173, 174.
Lexicographic row/column symmetry, 106–107, 177, 181, 274.
Lexicographically smallest (or largest) solution, 25–26, 111–113, 142, 157, 282, 283.
Lexicographically smallest traces, 84, 161, 162, 250.
Leyton-Brown, Kevin Eric, 125, 133.
Li, Chu Min (李初民), 131.
Li, Wei (李未), 149.
Lieb, Elliott Hershel, 251.
Life, Game of, 17–20, 97, 114, 137–139, 143, 167.
Light speed in Life, 139.
Line graph of a graph, 147, 249.
Linear equations, 26, 231.
Linear extensions, *see* Topological sorting.
Linear inequalities, 184.
  encoding of, 100–101, 172, 173.
Linear programming relaxation, 26.
Lines, abstracted, 106.
Links, dancing, 5, 121, 134, 208, 288.
Lisitsa, Alexei Petrovich (Лісіца, Аляксей Пятровіч), 281.
List coloring of graphs, 135, 151.
List merging, 231, 258.
Literal block distance, 72, 74, 158.
Literals, 2, 111.
  flushing, 76.
  internal representation, 28, 37, 66, 208, 209, 242, 257.
Litman, Ami (עמי ליטמן), 285.
Livelock, 22–23.
Llunell, Albert Oliveras i, 267.
LNCS: *Lecture Notes in Computer Science*, inaugurated in 1973.
Local Lemma, 81–90, 133, 151, 160–165.
Log encodings, 98, 114–115, 173.
Logemann, George Wahl, 31–32, 130, 298.
Longest simple path, 23, 203.
Lonlac, Jerry, 289.
Look-back, *see* Backjumping.
Lookahead autarky clauses, *see* Black and blue principle.
Lookahead forest, 42–44, 145–147, 168.
Lookahead solvers, 38–46, 55, 97, 129, 131, 176.
  combined with CDCL solvers, 129.
  compared to CDCL solvers, 98–100, 118–121, 182, 290.
Loopless shadows, 184.
Lopsidependency graphs, 82, 83, 160, 164, 165, 185, 224.
Lovász, László, 81, 82, 185, 191.
Loveland, Donald William, 32, 130, 298.
Lower semimodular lattices, 255–256.
Loyd, Samuel, 263.
Luby, Michael George, 80, 159.
Luks, Eugene Michael, 113.

Mμ: Megamems = millions of memory accesses, 69, 98.
Maaren, Hans van, 37, 46.
MacColl (= McColl), Hugh, 227.
MacMahon, Percy Alexander, Master Theorem, 250, 251.
Mader, Adolf, 275.
Madigan, Conor Francis, 132.
Magic, 193.
Magic sequences, 285.
Magnetic tape, 32.
Makespan, 172–173.
Mallach, Sven, 289.
Malik, Sharad (शरद मलिक), 132.
Maneva, Elitza Nikolaeva (Манева, Елица Николаева), 166, 256.
Mapping three items into two-bit codes, 179.
`march` solver, 40, 216.
Marek, Victor Wiktor, 216.
Markov (= Markoff), Andrei Andreevich (Марков, Андрей Андреевич), the elder, inequality, 158, 241.
Markov, Igor Leonidovich (Марков, Ігор Леонідович), 112, 281, 284.
Markström, Klas Jonas, 290.
Marques da Silva (= Marques-Silva), João Paulo, 132.
Marriage theorem, 224.
Martin, Alexander, 264.
Matching polynomial of a graph, 249.
Matchings in a graph: Sets of disjoint edges, 150, 230, 249.
  perfect, 109–110, 177.
Mathews, Edwin Lee (41), 67.
Matrix multiplication, 260.
Mauro, David Whittlesey, 192.
Maximal elements, 56, 62, 97, 115, 153, 157, 167.
Maximal planar graphs, 186.
Maximum independent sets, 87, 136, 187, 188.
Maximum number of 1s, 106–109, 135, 136, 177.
MAXSAT lower bound, 184.
“Maybe” state, 20.
Mazurkiewicz, Antoni Wiesław, 83.
McColl (= MacColl), Hugh, 227.
McGregor, William Charles, 7, 188.
  graphs, 7–8, 114–115, 134, 135, 188.
Mean running time, 120.

Median operation, 9, 136, 179.
Median running times, 99, 120–124, 127.
Megamem (Mμ): One million memory accesses, 201.
Méjean, Henri-Michel, 226.
Mellin, Robert Hjalmar, transforms, 151.
Mem (μ): One 64-bit memory access, 34.
MEM array, 66, 68, 124, 156.
Memo cache, 60, 233.
Memoization technique, 233.
Memoryless property, 244.
Menagerie, 116–117.
Merge networks, 266.
Merging lists, 231, 258.
Mertens, Stephan, 51.
Message passing, 90–95, 165–166.
Method I, 61.
Method IA, 61, 154.
Method of Trees, 129.
Methuselah solitaire, 282.
Methuselahs in Life, 19.
Mézard, Marc Jean Marcel, 51, 90, 91, 95.
Midpoint inequalities, 266.
Mijnders, Sid, 213.
Mills, Burton Everett, 130.
Minesweeper, 142–143.
Minimally unsatisfiable clauses, 150, 153.
Minimum covers, 193.
MiniSAT, 67.
Minterms, 179.
Mitchell, David Geoffrey, 50.
Miters, 121, 182.
Mitsche, Dieter Wilhelm, 51.
Mixed metaphors, 76.
Mixed-radix number systems, 268.
MMIX computer, ii, 158.
Mobile Life paths, 18–19, 138–139.
  flipflops, 138.
Möbius, Augustus Ferdinand, functions, 86.
  series, 86, 160, 162–163, 165, 247, 249.
Mod 3 addition, 114, 179.
Mod 3 parity, 179.
Mod 4 parity, 179.
Model checking, 16–17, 137–141, 179–180.
Model RB, 149.
Modified full adders, 114, 280.
Modular lattices, 255.
mone (-1), 242.
Monien, Burkhard, 215.
Monkey wrench principle, 113, 181.
Monotone functions, 163.
  Boolean, 137, 281.
Monotonic clauses, 5, 133, 157.
Monotonic paths, 108, 276.
Montanari, Andrea, 95.
Monus operation ($x \dot{-} y = \max\{0, x-y\}$), 92, 247, 268.
Moore, Edward Forrest, 202.
Morehead, Albert Hodges, 282.
Morel, Henri, 226.
Moser, Robin Alexander, 82, 254.
Moskewicz, Matthew Walter, 132.
Mossel, Elchanan (אלחנן מוסל), 166.
Mott-Smith, Geoffrey Arthur, 282.
Move codes, 29–31, 34, 144, 145, 155, 210.
MPR: Mathematical Preliminaries Redux, v.
Mueller, Rolf Karl, 60, 130.
Müller, Mike, 196.
Multicommodity flows, 170.
Multigraphs, 231.
Multilinear function, 86.
Multiplication of binary numbers, 8–9, 12–14, 114, 136, 173.
Multiplication of traces, 85, 161.
Multisets, 3, 214, 224, 250.
Multivalued graph colorings, 99.
Mutilated chessboard, 110, 114, 177–178, 286.
Mutual exclusion protocols, 20–24, 115, 139–141.
Mutzbauer, Otto, 275.
Mux operation (u? v: w), 81, 102, 152, 173, 219.
Mysterians, 290.

n-cube, 79, 136, 148, 184.
n.f.: not falsified, 271.
n queens problem, 25, 115, 171, 282.
NAND operation, 60.
Napier, John, Laird of Merchiston, 9, 173.
Near truth, 37–39.
Necessary assignments, 45, 146.
Negated auxiliary variables, 105.
Negative k-clauses, 157.
Negative literals, 2, 132, 153.
Nesting phase of lookahead, 40, 42–43, 145–147.
Newbie variables, 41.
Newton, Isaac, method for rootfinding, 216–217.
Niborski, Rodolfo, 190.
Niemelä, Ilkka Niilo Fredrik, 105.
Nieuwenhuis, Robert Lukas Mario, 267.
Nightingale, Peter William, 265.
No-player game, 17.
Nodes of a search tree, 34–35, 69, 124.
Noels, Alain, 202.
Noisy data, 181.
Nonattacking queens, 25, 115, 171, 282.
Nonaveraging sets, 114, 135.
Nonchromatic rectangles, 176–177.
Nonchronological backtracking, *see* Backjumping.
Noncommutative variables, 162.
Nonconstructive proofs, 57, 58, 81, 202.
Nondeterministic finite-state automata, 175.
Nondeterministic polynomial time, 131.
Nondeterministic processes, 20, 141, 182.
Nonintersecting paths, 170.

Nonnegative coefficients, 164.
Nonprimary columns, 186.
Nonterminal symbols, 175.
Normal chains, 278.
Normal functions, 279.
Not-all-equal SAT, 185.
Notational conventions, vi.
  ∂S (boundary set), 58, 154, 180, 188.
  $C' \diamond C''$ (resolvent), 54, 152.
  $C \subseteq C'$ (subsumption), 61, 152.
  $F \,|\, l$ (F given l), 27, 96, 291.
  $F \,|\, L$ (F given L), 27, 103, 157.
  $F \vdash C$ (F implies C), 59, 152, 153.
  $F \vdash_1 \epsilon$, 70, 157, 175.
  $F \vdash_1 l$, 103–104, 176.
  $F \vdash_k \epsilon$,
  $F \vdash_k l$, 175–176.
  $G \oplus H$ (direct sum), 162, 177.
  $|l|$ (a literal's variable), 2.
  $\pm v$ (v or $\bar{v}$), 2.
  $\langle xyz \rangle$ (median), 9.
  $x \,\&\, y$ (bitwise AND), 28, 29, 31, 37, 38, 66, 68, 76, 81, 196, 209–211, 220, 241.
  $x \mid y$ (bitwise OR), 43, 196, 241, 258–259.
  $x \oplus y$ (bitwise XOR), 28, 137, 196, 208, 220, 241.
  $x \dot{-} y$ (monus), 92, 247, 268.
  $x?\ y\colon z$ (if-then-else), 81, 102, 152, 173, 219.
  $w(\alpha)$, 57.
  $w(\alpha \vdash \epsilon)$, 57.
  $\|\alpha \vdash C\|$, 57.
  $\mu(C)$, 59, 153.
Novikov, Yakov Andreevich (Новиков, Яков Андреевич), 70, 132.
Nowakowski, Richard Joseph, 107, 275.
NP-complete problems, 1, 3, 27, 87, 130–131, 134, 142, 151, 181–183, 207, 268, *see also* CoNP-complete problems.
NT (near truth), 37–39.
Null clause (ϵ), 3, 27, 185, 291.
Null list, representation of, 33, 210.
Null partial assignment, 166.
Null set ($\emptyset$), 185.
Null string (ϵ), 85.
Nullary clause (ϵ), 3, 27, 185, 291.
Number theory, 14, 137, 192.

Occurrence threshold of a graph, 160.
Odd permutations, 218.
Odd-even merge network, 266.
Odd-even transposition sort, 263.
Oliveras i Llunell, Albert, 267.
On-the-fly subsumptions, 124, 156.
One-in-three satisfiability, 183.
One-per-clause satisfiability, 183.
Open shop scheduling problems, 115, 172–173.
OR operation, 9, 10, 13, 258.
  bitwise ($x \mid y$), 43, 196, 241, 258–259.
Orbits of a permutation group, 108, 277.
Order encoding, 98–101, 114, 120, 170–173, 190, 268, 281.
Order of a permutation, 111.
Organ-pipe permutations, 171.
Oriented cycle detection, 260.
Oriented trees, 108.
Orphan patterns in Life, 139.
Orponen, Olli Pekka, 80.
Oscillators in Life, 19, 138–139.
Output states, 175.
OVAL array, 74, 125, 237, 240.
Overfitting, 182.
Overflow in arithmetic, 67, 240.
Oxusoff, Laurent, 215.

$\wp$ (tautology, the always-true clause), 3, 58, 60, 152, 180, 215, 226–228, 258.
P = NP, 1.
Palindromes, 136.
Panagiotou, Konstantinos (Παναγιώτου, Κονσταντίνος), 221.
Papadimitriou, Christos Harilaos (Παπαδημητρίου, Χρίστος Χαριλάου), 77, 240, 241.
Parallel multiplication circuits, 12–14, 137.
Parallel processes, 20, 24, 121, 128–129.
Parameters, tuning of, 80–81, 93–94, 124–128.
ParamILS, 125, 286–287.
Parity-related clauses, 153–154, 172, 178, 231–232, 290.
Partial assignments, 30, 61, 62, 165, 176.
Partial backtracking, 208.
Partial latin square construction, 151.
Partial orderings, 56, 85, 115, 248.
  of dimension ≤ 2, 213.
Participants, 41, 44, 145.
Path detection, 169.
Path graphs P_n, 84, 160, 253.
Patience, *see* Solitaire games.
Paturi, Ramamohan (రామమోహన్ పాతూరి), 288.
Paul, Jerome Larson, 5.
Paull, Marvin Cohen, 148.
PC_k, 176, 178.
Pearl, Judea (יהודה פרל), 95.
Pegden, Wesley Alden, v, 164, 253.
Peierls, Rudolf Ernst, 95.
Peres, Yuval (יובל פרס), 221.
Pérez Giménez, Xavier, 51.
Perfect matchings in a graph, 109–110, 177.
Permanent of a matrix, 183, 251.
Permutation polynomial of a set, 163.
Permutation posets, 213.
Permutations, 105, 265.
  signed, *see* Signed permutations.
  weighted, 163.
Permuting variables and/or complementing them, *see* Signed permutations.

Peterson, Gary Lynn, 23, 115, 140, 204.
Petrie, Karen Elizabeth Jefferson, 283.
Phase saving, 67, 75.
Phase transitions, 50–52, 149–150.
Phi (ϕ), 146, 147, 160, 251.
Phoenix in Life, 198, 207.
Pi (π), as source of “random” data, 12, 46, 108, 115, 147, 184, 193, 286; *see also* Pi function.
Pi function, 102, 174.
Pieces, in trace theory, 84–87.
Pigeonhole principle, 57.
  clauses for, 57–59, 105–106, 113, 153, 176, 181, 186, 265.
Pikhurko, Oleg Bohdan (Піхурко, Олег Богданович), 285.
Pile sums, 151.
Pincusians, 133.
Pipatsrisawat, Thammanit (= Knot) (พิพัฒน์ศรีสวัสดิ์, ธรรมนิติ์ (= นอต)), 67, 262.
Pixel images, 200; *see also* Grid patterns.
Plaisted, David Alan, 102.
Planning, 132.
Playing cards, 114, 180, 282.
Points, abstracted, 106.
Poison cards, 282.
Poisson, Siméon Denis, probability, 225.
Polarities, 3, 67, 76, 207, 237.
Pólya, György (= George), theorem, 284.
Polynomials in trace theory, 85.
Population in Life, 19.
Portfolio solvers, 133.
Posets, *see* Partial orderings.
Positive autarkies, 146.
Positive j-clauses, 157.
Positive literals, 2, 132, 146.
Posthoff, Christian, 275.
Postorder, 42–43, 214.
Postprocessor, 96.
Preclusion clauses, 99, 171, 186.
Preorder, 42–43, 214.
Preprocessing of clauses, 95–97, 103, 166–168, 182, 268, 272, 278.
Preselection phase of lookahead, 40–42, 147.
Prestwich, Steven David, 264.
Primary variables, 104, 105.
Prime clauses, 174, 270, 273.
Prime implicants, 281.
Pringsheim, Alfred Israel, 88, 164.
Prins, Jan Fokko, 267.
Probabilistic method, 81.
Probability of satisfiability, 47–54.
$prod(m,n)$, 12–14, 114, 137.
Production rules, 175.
Profile of a search tree, 151.
Progress, display of, 30, 145, 155.
Progress saving, 67, *see* Phase saving.
Projection of a path, 184.
Projective plane, 274.
Propagation, kth order, 175–176, 273.
Propagation completeness (UC_1), 176.
Proper ancestors, 164.
Proto truth, 37, 42.
Prover–Delayer game, 55–56, 152–153.
PSATO solver, 159.
Pseudo-Boolean constraints, *see* Threshold functions.
`PT` (proto truth), 37, 42.
Pudlák, Pavel, 55.
Puget, Jean-François, 113.
Purdom, Paul Walton, Jr., 30, 32, 131, 151, 226.
Pure cycles, 140.
Pure literals, 29, 31, 32, 34, 44, 60, 130, 135, 146, 152, 208, 215, 227, 256, 259, 268, 269, 275.
Purging unhelpful clauses, 68, 71–75, 124, 132, 157, 158, 168, 182, 184, 235.
  threshold for, 74, 125, 127.
Putnam, Hilary, 9, 32, 130, 298.
Pyramids in trace theory, 87, 162.

q.s.: quite surely, 149, 153, 169.
QDD: A quasi-BDD, 188.
Quad-free matrices, 106–107, 113, 176–177, 274, 284.
Quantified formulas, 60, 154.
Queen graphs, 25, 99–100, 114–115, 120, 171, 180, 181.
Quenchable graphs, 179–180, 281.
Quick, Jonathan Horatio, 181.
Quilt patterns, 198.
Quimper, Claude-Guy, 272.
Quine, Willard Van Orman, 129, 130.

$\mathcal{R}(G)$ (Local Lemma bounds), 82, 87–90, 160, 163–165.
Radio colorings, 136.
Radix-d representation, 173.
Rado, Richard, 191.
Ramakrishnan, Kajamalai Gopalaswamy, 16.
raman graphs, 231.
Ramani, Arathi (ஆர்த்தி ரமணி), 112, 281, 284.
Ramanujan Iyengar, Srinivasa (ஸ்ரீனிவாஸ ராமானுஜன் ஐயங்கார்), graphs, 154; *see also raman* graphs.
Ramos, Antonio, 75.
Ramsey, Frank Plumpton, theorem, 81.
rand, 39–40, 46, 50, 115, 147, 182.
Random bits, biased, 12, 241.
Random choices, 12.
Random decision variables, 125–127, 155, 286.
Random graphs, 81.
Random permutations, 233.

Random satisfiability problems, 47–54, 91, 151.
  2SAT, 51–54, 149.
  3SAT, 39–40, 46–51, 59–60, 80, 93–94, 147–149, 153, 242.
  kSAT, 49–51, 146, 148.
Random walks, 77–81, 125, 243.
Random words, 149.
Randomized methods, 77, 129, 182, 210.
`RANGE` scores, 74, 125–127, 158, 239.
RAT, *see* Resolution certifiable clauses.
Rauzy, Antoine Bertrand, 131, 215.
Reachability in a graph, 169.
Ready list, 32.
Real roots of polynomials, 163, 249.
Real truth, 37–39.
Reasons, 62, 72, 157, 165, 233.
Rebooting, 22.
Reckhow, Robert Allen, 61.
Recurrence relations, 151, 177, 189, 215, 243.
Recursive procedures, 27, 130, 172, 186, 233.
Recycling of clauses, 66, 124.
Reduction of clauses, 27, 143; *see also* Simplification of clauses.
Redundant clauses, 257.
Redundant literals, 65, 155–156, 232, 234.
Redundant representations, 171.
Reed, Bruce Alan, 52.
Reflected ternary code, 290.
Reflection symmetries, 112, 138, 156.
Refutation chains, 57, 227.
Refutation trees, 152.
Refutations, 54–60, 70, 110, 152; *see also* Certificates of unsatisfiability.
Regular expressions, 174–175.
Regular resolution, 55, 152, 231.
Reinforcement messages, 91–93.
Reliability polynomials, 83.
Reluctant doubling, 77, 80–81, 159–160.
Reluctant Fibonacci sequence, 160.
Renamed Horn clauses, 176, 263.
Repeated clauses, 49.
Replacement principle, 96.
Representation of Boolean functions, 104, *see* Encoding into clauses.
Representing three states with two bits, 179.
Rescaled activity scores, 67.
Resende, *see* Guilherme De Carvalho Resende.
Resizing of data structures, 210.
Resolution certifiable clauses, 261.
Resolution chains, 57–59, 152, 153, 227.
Resolution of clauses, 54–65, 70, 101, 129, 130, 144, 167, 185, 215, 224, 256.
  implementation of, 167.
Resolution refutations, 54–60, 70, 110, 152; *see also* Certificates of unsatisfiability.
  extended, 60, 71, 133, 154, 168, 215.
  regular, 55, 152, 231.
  treelike, 55–56, 152–153.
Resolvable clauses, 164.
Resolvent ($C' \diamond C''$), 54, 130, 152.
Restarting, 80–81, 95, 125, 132.
  and flushing literals, 68, 75–77, 124, 132, 157, 158, 169, 234, 246.
Restricted growth strings, 179.
Restricted pigeonhole principle, 58.
Reusing the trail, 75.
Reverse unit propagation, 71.
Revolving door Gray code, 282.
Reynaud, Gérard, 226.
Richards, Keith, 1.
Rickard, John, 290.
Right division of traces, 85, 161.
Right factor of a trace, 161.
Riis, Søren Møller, 110.
Ripoff, Robert Iosifovich (Рипов, Роберт Иосифович), 7.
Rivest, Ronald Linn, clauses, 4, 55, 70, 134, 144, 182.
Roberts, Fred Stephen, 136.
Robertson, Aaron Jon, 185.
Robinson, Gilbert de Beauregard, 275.
Robinson, John Alan, 59, 96, 227.
Rodríguez Carbonell, Enric, 267.
Rokicki, Tomas Gerhard, 200.
Rooij, Iris van, 207.
Rook paths, 206.
Rookwise connected cells, 170.
Ross, Kenneth Andrew, 282.
Rotational symmetry, 138, 202, 275.
Rotors in Life, 138.
Roussel, Olivier Michel Joseph, 132, 272.
Routing, disjoint, 170.
Row sums, 151.
Roy, Amitabha (অমিতাভ রায়), 113.
`RT` (real truth), 37–39, 43.
Ruler doubling, 160.
Ruler of Fibonaccis, 246.
Running times, 89–90.
  comparison of, 34–35, 39, 69, 97–100, 105–107, 110, 112, 118–128, 182, 184, 218, 237, 264, 281, 290.
  mean versus median, 120.
  worst case, 144, 146, 154.
Runs of 1s, 26, 143, 175.

s-chains, 52–53, 149.
s-snares, 53, 149.
$S_1(y_1, \ldots, y_p)$, 6.
$S_k(m,n)$, 50–54.
$S_{k,n}$, 49–51, 148, 149.
$S_{\leq r}(x_1, \ldots, x_n)$ and $S_{\geq r}(x_1, \ldots, x_n)$, 8, *see* Cardinality constraints.
Saddle point method, 226.
Sahni, Sartaj Kumar (सरताज कुमार साहनी), 268.
Saïs, Lakhdar (ⵙⴰⵢⵙ ⵍⴰⵅⴹⴰⵔ, سايس لخضر), 236, 289.

Sakallah, Karem Ahmad (أحمد ساق الله كارم), 112, 132, 281, 284.
Salhi, Yakoub (يعقوب صالحي), 289.
Sampling with and without replacement, 49–50, 132, 226.
Samson, Edward Walter, 60, 130.
SAT: The satisfiability problem, 3.
SAT solvers, 1, 131–133.
SATexamples.tgz, 118.
Satisfiable formulas, 1.
  variability in performance, 35, 120–121, 128, 287.
Satisfiability, 1–184.
  history, 32, 59–60, 105, 129–133.
  thresholds for, 50–54, 91, 148–149, 221.
Satisfiability-preserving transformations, 107–113.
Satisfying assignments, 1, 30, 143–144, 166, 214, 219.
SATzilla solver, 132–133.
Schaefer, Thomas Jerome, 289.
Schensted, Craige Eugene (= Ea Ea), 275.
Schlipf, John Stewart, 274.
Schmitt, John Roger, 285.
Schoenfield, Jon Ellis, 192.
Schöning, Uwe, 78.
Schrag, Robert Carl, 132.
Schroeppel, Richard Crabtree, 197.
Schwarzkopf, Bernd, 282.
Scott, Alexander David, 224, 251, 252.
Scott, Allan Edward Jolicoeur, 207.
Scott, Sidney Harbron, 191.
Scoville, Richard Arthur, 162.
Search trees, 28–29, 32–34, 124, 152.
  expected size, 151–152.
  optimum, 144.
Second moment principle, 54, 221, 222.
Seitz, Simo Sakari, 80.
Self-subsumption, 96, 167, 168, 257.
Selman, Bart, 50, 79, 132.
Semimodular lattices, 255–256.
Sentinel values, 259.
Sequential consistency, 24.
Sequential lists, 36–37, 144.
Sequents, 59.
Serial correlation coefficients, 143.
Set partitions, 220.
SGB, *see* Stanford GraphBase.
Shadows of paths, 184.
Shandy, Tristram, iii.
Sharp thresholds, 51–52, 149.
Shearer, James Bergheim, 82, 87, 160.
Sheeran, Mary, 203.
Shlyakhter, Ilya Alexander (Шляхтер, Илья Александрович), 284.
Shmoys, David Bernard, 267.
Shortest paths, 262.
Shortz, William Frederic, v.
SIAM: The Society for Industrial and Applied Mathematics, 204.
Sideways sum (νx): Sum of binary digits, 114, 143, 179, 195, 279.
  second order ($\nu^{(2)}x$), 143.
Sifting, 219, 220.
Siftup in a heap, 234.
Signature of a clause, 72, 158.
Signature of a literal, 258.
Signed mappings, 180–181.
Signed permutations, 4, 111, 178.
  involutions, 112–113, 180, 277–278.
Silva, *see* Marques da Silva.
Silver, Stephen Andrew, 138, 200.
Simmons, Gustavus James, 192.
Simon, Laurent Dominique, 72, 132.
Simple cycles and paths, 23–24, 140.
simplex graphs, 136.
Simplification of clauses, 65, 155, 232; *see also* Preprocessing of clauses.
Sims, Charles Coffin, tables, 283.
Simultaneous read/write, 141.
Simultaneous write/write, 141.
Sinclair, Alistair, 80, 159, 256.
Singh, Satnam, 203.
Single lookahead unit resolution, 105, 176.
Single-stuck-at faults, 10–14, 114, 136–137.
Sink: A vertex with no successor, 87, 214.
  components, 108–110.
Sinz, Carsten Michael, v, 8, 117, 118, 135, 174, 189, 280.
Skip Two solitaire, 282.
Slack, in trace theory, 88, 251.
Slisenko (= Slissenko), Anatol Olesievitch (Слисенко, Анатоль Олесьевич), 59.
SLS: Stochastic local search, 77.
SLUR algorithm, 105, 176.
Sly, Allan Murray, 51.
Smile, 207.
Smith, Barbara Mary, 283.
Snake dance, 138.
Snakes, 52–54, 149.
Snares, 52–54, 149.
Snark graphs, 69, 147, 153, 157.
Snevily, Hunter Saint Clair, 5.
Socrates, son of Sophroniscus of Alopece (Σωκράτης Σωφρωνίσκου Ἀλωπεκῆθεν), 129.
Soft clauses, 168.
Sokal, Alan David, 251, 252.
Solitaire games, 180, 282.
Solutions, number of, 48, 219.
Somenzi, Fabio, 236.
Sörensson, Niklas Kristofer, v, 67, 155, 203, 268.
Sorting networks, 115, 137, 203, 263, 266.
Source: A vertex with no predecessor, 87, 252.
Spaceships in Life, 139, 201.

Spanning trees, 281, 290.
Sparse encoding, *see* Direct encoding.
Speckenmeyer, Ewald, 131, 215.
Spence, Ivor Thomas Arthur, 290.
Spencer, Joel Harold, 81, 82, 254.
Spiral order, 206.
Stable Life configurations, 19, 197.
Stable partial assignments, 165–166.
Stacks, 37–39, 43.
Stacking the pieces, 84–85.
Stålmarck, Gunnar Martin Natanael, 56, 132, 153, 203, 232, 238.
Stamm-Wilbrandt, Hermann, 131.
`STAMP(l)`, 258.
Stamping of data, 37–38, 64, 66, 145, 155, 211, 236, 258–260.
Standard deviation, 48, 240.
Stanford GraphBase, ii, 12, 13, 126, 179, 214, 231.
Stanford University, 282.
Stanley, Richard Peter, 275.
Starfish graphs, 249.
Starvation, 22–24, 115, 140, 141.
Statistical mechanics, 90.
Stators in Life, 138.
Stege, Ulrike, 207.
Stein, Clifford Seth, 267.
Steinbach, Heinz Bernd, 275.
Steiner, Jacob, tree packing, 264.
  triple systems, 106, 274.
Sterne, Laurence, iii.
Stickel, Mark Edward, 132.
Sticking values, 67, *see* Phase saving.
Still Life, 19, 138, 200.
Stirling, James, approximation, 221, 240.
  subset numbers, 149, 220, 226.
Stochastic local search, 77.
Stopping time, 48–50, 148.
Strahler, Arthur Newell, numbers, 152.
Strengthening a clause, 96, 156, 259–260.
Stříbrná, Jitka, 224.
Strichman, Ofer (עופר שטריכמן), 203.
Strictly distinct literals, 2–3, 52, 165.
Strings generalized to traces, 83.
Strong components: Strongly connected components, 41–42, 52–53, 108, 131, 215, 221, 263.
Strong exponential time hypothesis, 183.
Strong product of graphs, 134.
Strongly balanced sequences, 179.
Stuck-at faults, single, 10–14, 114, 136–137.
Stützle, Thomas Günter, 125.
Subadditive law, 59.
Subcubes, 148.
Subforests, 42.
Subinterval constraints, 190.
Submatrices, 106–109, 177.
Subset sum problem, 268.
Substitution, 257.
Subsumption of clauses, 61, 96, 124, 152, 155, 156, 166–168, 181, 269.
  implementation, 167, 259.
  on-the-fly, 124, 156.
Subtraction, encoding of, 100.
Sudoku, 183, 225.
Summation by parts, 48.
Summers, Jason Edward, 200.
Sun, Nike (孙妮克), 51.
Support clauses, 99, 114, 171.
Survey propagation, 51, 90–95, 165–166, 213.
Swaminathan, Ramasubramanian (= Ram) Pattu (ராமசுப்ரமணியன் பட்டு சவாமிநாதன்), 274.
Swapping to the front, 211, 242.
Sweep of a matrix, 108–109, 177.
Swoop of a matrix problem, 109.
Syllogisms, 129.
Symeater in Life, 200.
Symmetric Boolean functions, 179, 207, 219, 270; *see also* Cardinality constraints.
  $S_{\le 1}$, *see* At-most-one constraint.
  S_1, 6, 220.
  $S_{\ge 1}$, *see* At-least-one constraint.
  S_r, 135, 179, 256.
Symmetric threshold functions, *see* Cardinality constraints.
Symmetrical clauses, 105–106, 156.
Symmetrical solutions, 138, 183, 274.
Symmetries of Boolean functions, 178.
Symmetry breaking, vii, 5, 105–114, 138, 176–181, 187, 188, 190–192, 238, 267, 281–283, 285, 288–290.
  in graph coloring, 99–100, 114, 171, 179, 187.
Symmetry from asymmetry, 19, 201.
Synthesis of Boolean functions, 137, 178–179, 194.
Szabó, Tibor András, 224.
Szegedy, Márió, 90, 161, 255.
Szeider, Stefan Hans, 224, 284.
Szemerédi, Endre, 59.
Szpankowski, Wojciech, 225.

t-snakes, 53, 54, 149.
Tμ: teramems = trillions of memory accesses, 110, 121, 126, 265, 281.
Tableaux, 275.
Taga, Akiko (多賀明子), 264, 267.
Tajima, Hiroshi (田島宏史), 100.
Tak, Peter van der, 75.
Takaki, Kazuya (高木和哉), 224.
Tamura, Naoyuki (田村直之), 100, 171, 264, 267, 268.
"Take account," 37, 43, 45–46, 217, 235.
Tanjo, Tomoya (丹生智也), 268.
TAOCP: *The Art of Computer Programming*, problem, 115, 169.
Tape records, 32.
Tardos, Gábor, 82, 224, 254.

Tarjan, Robert Endre, 41, 42, 214, 217.
Tarnished wires, 13, 193.
Tatami tilings, 115, 143.
TAUT: The tautology problem, 3, 129, 130.
Tautological clause (℘), 3, 58, 60, 152, 180, 215, 226–228, 258.
Tensors, 151.
Teramem (Tμ): One trillion memory accesses, 40, 106, 107, 110, 217, 218, 286.
Ternary clauses, 3–6, 36, 118, 131, 183; *see also* 3SAT.
Ternary numbers, 100, 141, 179.
Ternary operations, 9, 136.
Territory sets, 84, 161, 163.
Test cases, 113–124.
  capsule summaries, 114–115.
Test patterns, *see* Fault testing.
Tetris, 84.
Theobald, Gavin Alexander, 190.
Theory and practice, 109.
Three-coloring problems, *see* Flower snarks.
Threshold functions, 100–101, 175.
Threshold of satisfiability, 50–54, 91, 148–149, 221.
Threshold parameter Θ, 126, 213, 286.
Thurley, Marc, 262.
Tie-breakers, 74, 239.
Tiling a floor, 115, 138, 143, 199.
Time stamps, *see* Stamping of data.
Timeouts, 120.
TIMP tables, 36–40, 43, 45, 144–145.
To-do stack, 259.
Tomographically balanced matrices, 141.
Tomography, 24–26, 115, 141–143, 167, 285.
Top-down algorithms, 252.
Topological sorting, 85, 248.
Toruses, 134, 138, 200.
Touched clauses, 44.
Touched variables, 259.
Tovey, Craig Aaron, 150, 223.
Tower of Babel solitaire, 282.
Tower of London solitaire, 282.
Trace of a matrix: The sum of its diagonal elements, 108, 218.
Traces (generalized strings), 83–90, 161–162, 252, 254.
Tradeoffs, 125–126.
Trail (a basic data structure for Algorithm C), 62–65, 68, 72, 124, 166, 236, 238.
  reusing, 75.
Training sets, 15–16, 115, 125–127, 133, 137, 182, 286.
Transitions between states, 16–24, 175, 202, 218.
Transitive law, 56, 228.
Tree-based lookahead, *see* Lookahead forest.
Tree function, 230.
Tree-ordered graphs, 163–164.
Treelike resolution, 55–56, 152–153.
Treengeling solver, 121.
Triangle-free graphs, 167.
Triangles (3-cliques), 167, 238, 264.
Triangular grids, 136.
Tribonacci numbers, 216.
Triggers, 46, 126.
Trivalent graphs, 147, 154, 231.
Trivial clauses, 124–127, 156, 236, 239.
Trivially satisfiable clauses, 3.
Truemper, Klaus, 273.
Truszczyński, Mirosław (= Mirek) Janusz, 216.
Truth, degrees of, 37–39, 42–43, 45–46, 216.
Truth tables, 129–130, 179, 194, 220, 277.
Tseytin, Gregory Samuelovich (Цейтин, Григорий Цамуилович), 9, 59–60, 71, 133, 152, 154, 168, 178, 215, 231, 290.
  encodings, 9, 17, 101–102, 136, 173, 195.
  encodings, half of, 192, 268.
Tsimelzon, Mark Boris, 134.
Tuning of parameters, 124–128, 133, 182.
Turán, Pál (= Paul), 190.
Turton, William Harry, 180.
Two-level circuit minimization, 257.

UC_k, 176, 273.
UIP: Unique implication point, 132, 233.
Unary clauses, *see* Unit clauses.
Unary representation (= order encoding), 98–101, 114, 120, 170–173, 190, 268, 281.
Undoing, 28–31, 37–39, 95–96, 143–145, 208, 212, 217–218.
Uniform distribution, 159.
Unique implication points, 132, 233.
Uniquely satisfiable clauses, 48, 219.
Unit clauses (= unary clauses), 3, 6, 9, 13, 21, 23, 30, 31, 33, 35, 36, 66, 70, 130, 144, 151, 157, 192, 205, 210, 238, 290.
Unit conditioning, 27, 96, 166, 259, 261.
Unit propagation ($\vdash_1$), 31–34, 36, 62, 65, 68, 70–71, 93, 97–99, 103–104, 132, 155, 157, 165, 171, 174, 236, 270, 272, 276.
  generalized to $\vdash_k$, 175.
Universality of Life, 17.
Unnecessary branches, 55, 227.
Unsatisfiable core, 185.
Unsatisfiable formulas, 1.
  implications of, 104, 175–176.
Unsolvable problems, 130.
Urns and balls, 221.
Urquhart, Alisdair Ian Fenton, 231.

VAL array, in Algorithm C, 66–68, 73–76, 233–236, 238, 240.
  in Algorithm L, 37–39, 43, 216.
Valid partial assignments, 165–166.
Van de Graaff, Robert Jemison, 198.
van der Tak, Peter, 75.

van der Waerden, Bartel Leendert, 4.
numbers, 5, *see* $W(k_0,\ldots,k_{b-1})$.
van Deventer, Mattijs Oskar, 290.
Van Gelder, Allen, 71, 233, 237, 263.
van Maaren, Hans, 37, 46.
van Rooij, Iris, 207.
van Zwieten, Joris Edward, 37.
VAR array, in Algorithm L, 38, 182, 211.
Variability in performance on satisfiable problems, 35, 120–121, 128, 287.
on unsatisfiable problems, 69, 121, 128, 287.
Variable elimination, 96–97, 101, 102, 129, 154–155, 166–168, 173, 174, 256–257, 259–260, 270, 272.
Variable interaction graphs, 116–118, 182.
Variables, 2.
introducing new, 3, 6, 8, 9, 13, 60; *see* Auxiliary variables, Extended resolution.
Variance, 49, 158, 164, 240, 243.
Vassilevska Williams, Virginia Panayotova (Василевска, Виргиния Панайотова), 167.
Vaughan, Theresa Phillips, 162.
Verification, 16, 157; *see also* Certificates of unsatisfiability.
Viennot, Gérard Michel François Xavier, 83, 84, 87, 162, 249.
Vinci, Leonardo di ser Piero da, 7.
Virtual unswapping, 211.
Visualizations, 116–118.
Vitushinskiy, Pavel Viktorovich (Витушинский, Павел Викторович), 282.
Vries, Sven de, 206.
VSIDS, 132.

$W(k_0,\ldots,k_{b-1})$ (van der Waerden numbers), 4–5, 127, 133.
$waerden(j,k;n)$, 4–5, 32, 35, 37, 39–42, 45, 63–66, 69, 71–75, 97, 112, 115, 121, 127–129, 133, 142–145, 156, 157, 166, 167, 181, 210, 236, 256.
Waerden, Bartel Leendert van der, 4.
numbers, 5, *see* $W(k_0,\ldots,k_{b-1})$.
Wagstaff, Samuel Standfield, Jr., 190.
Wainwright, Robert Thomas, 138, 166, 197, 198.
Walks in a graph, 260.
WalkSAT algorithm, 79–81, 93–94, 118, 125, 159–160, 182, 191, 265, 281.
Walsh, Toby, 272.
Warmup runs, 125, 239.
Warners, Johannes (= Joost) Pieter, 268.
Warrington, Gregory Saunders, 285.
Watched literals, 30–34, 65–66, 68, 132, 144, 155, 233–236.
Weakly forcing, 174.
Websites, ii, iii, v, 118.
Weighted permutations, 163.
Wein, Joel Martin, 267.
Weismantel, Robert, 264.
Welzl, Emmerich Oskar Roman (= Emo), 158.
Wermuth, Udo Wilhelm Emil, v.
Wetzler, Nathan David, 71, 239.
Wheel graphs (W_n), 191.
Whittlesey, Marshall Andrew, 192.
Width of a resolution chain, 57–59, 153–154.
Wieringa, Siert, 129.
Wigderson, Avi (אבי ויגדרזון), 57–58, 153, 231.
Wilde, Boris de, 213.
Williams, Richard Ryan, v, 270.
Williams, Virginia Panayotova Vassilevska (Виргиния Панайотова Василевска), 167.
Wilson, David Bruce, 54, 149, 221.
Windfalls, 43, 147, 182, 217.
Winkler, Peter Mann, 290.
Winn, John Arthur, Jr., 275.
Wires of a circuit, 10–14, 136.
Wobble function, 51, 151.
Worst case, 144, 146, 154, 239, 244.
Write buffers, 24.

Xeon computer, 289.
XOR operation, 9, 10, 13, 136.
bitwise ($x \oplus y$), 28, 137, 196, 208, 220, 241.
Xray-like projections, 24.
Xu, Ke (许可), 149.
Xu, Lin (徐林), 133.
Xu, Yixin (徐一新), 255.

Yaroslavtsev, Grigory Nikolaevich (Ярославцев, Григорий Николаевич), 280.
Yeh, Roger Kwan-Ching (葉光清), 192.
Yuster, Raphael (רפאל יוסטר), 260.

$Z(m,n)$ (Zarankewicz numbers), 106–107, 176.
Zanette, Arrigo, 206.
Zarankiewicz, Kazimierz, 106.
quad-free problem, 106–107, 113, 176.
Závodný, Jakub, 196.
Zecchina, Riccardo, 51, 90, 91, 256.
Zhang, Hantao (张瀚涛), 129, 132.
Zhang, Lintao (张霖涛), 132.
Zhao, Ying (赵颖), 132.
Zhu, Yunshan (朱允山), 132.
ZSEV (zero or set if even), 242.
Zuckerman, David Isaac, 80, 159.
Zwick, Uri (אורי צוויק), 260.
Zwieten, Joris Edward van, 37.